Uniwersytet Mikołaja Kopernika
Wydział Fizyki, Astronomii
i Informatyki Stosowanej
Instytut Fizyki

Krzysztof Jackowski
nr albumu: 199556

Praca inżynierska
na kierunku fizyka techniczna

Aplety komputerowe w indukcji elektromagnetycznej

Opiekun pracy dyplomowej
Prof. zw. dr hab. inż. Grzegorz Karwasz
Zakład Dydaktyki Fizyki

Toruń 2010

Pracę przyjmuję i akceptuję

.....

data i podpis opiekuna pracy

Potwierdzam złożenie pracy dyplomowej

.....

data i podpis pracownika dziekanatu

10, kursywa, Na odwrotnej stronie strony z podziękowaniami powinien pojawić się tekst:

UMK zastrzega sobie prawo własności niniejszej pracy inżynierskiej w celu udostępniania dla potrzeb działalności naukowo-badawczej lub dydaktycznej

Spis treści

1. Wstęp 5
2. Oprogramowanie i język 6
2.1. Java 6
2.2. Netbeans 12
2.3. Microsoft Office Publisher 2003 13
3. Indukcja elektromagnetyczna i prezentacja tego zjawiska za pomocą apletów.. 14
3.1. Indukcja elektromagnetyczna – definicja 14
3.2. Aplety komputerowe jako sposób przedstawiania zjawiska indukcji elektromagnetycznej. 18
4. Definicja elektrostatyki i jej krótka prezentacja za pomocą apletu. 27
4.1. Elektrostatyka – definicja 27
4.2. Elektryzowanie przez tarcie – aplet 29
5. Aplety komputerowe. 30
5.1. Wsuwanie i wysuwanie magnesu. 30
5.2. Wsuwanie i wysuwanie elektromagnesu. 42
5.3. Włączanie i wyłączanie prądu w zwojnicy elektromagnesu. 46
5.4. Elektromagnes z opornicą suwakową. 50
5.5. Ramka wchodząca w pole magnetyczne. 54
5.6. Ramka obracająca się w polu magnetycznym. 67
5.7. Elektryzowanie przez tarcie. 78
6. Podsumowanie 88
7. Literatura 89
8. Załączniki 90
8.1. Załącznik 1. 90
8.2. Załącznik 2. 92
8.3. Załącznik 3. 95
8.4. Załącznik 4. 98
8.5. Załącznik 5. 101
8.6. Załącznik 6. 118
8.7. Załącznik 7. 123

1. Wstęp

Niniejsza praca jest pracą inżynierską napisaną pod opieką prof. dr hab. Grzegorza Karwasza. Sam temat pracy „Aplety komputerowe w indukcji elektromagnetycznej” wskazuje, że nie jest to praca czysto fizyczna, ale prezentuje również sposób, w jaki można przedstawić wybrane problemy fizyczne za pomocą symulacji komputerowych, którymi są w tej pracy aplety. Praca jest podzielona na sześć podstawowych rozdziałów.

Rozdział pierwszy w skrócie przedstawia tematykę kolejnych rozdziałów.

W drugim rozdziale oraz jego podrozdziałach opisuję język programowania i środowisko, w jakim zostały stworzone i przedstawione owe animacje.

Rozdział trzeci ukazuje istotę indukcji elektromagnetycznej oraz wyjaśnia podstawowe zagadnienia fizyczne prezentowane przez aplety.

W rozdziale czwartym przedstawiam krótko pojęcie elektrostatyki i jeden przykładowy aplet z nią związany.

Rozdział piąty w szerokim zakresie opisuje aplety od strony programistycznej. Znajduje się w nich szczegółowa analiza kodów źródłowych stworzonych przeze mnie apletów.

W podsumowaniu niniejszej pracy zamieściłem kilka drobnych uwag dotyczących trudności pojawiających się podczas jej tworzenia.

2. Oprogramowanie i język

2.1. Java

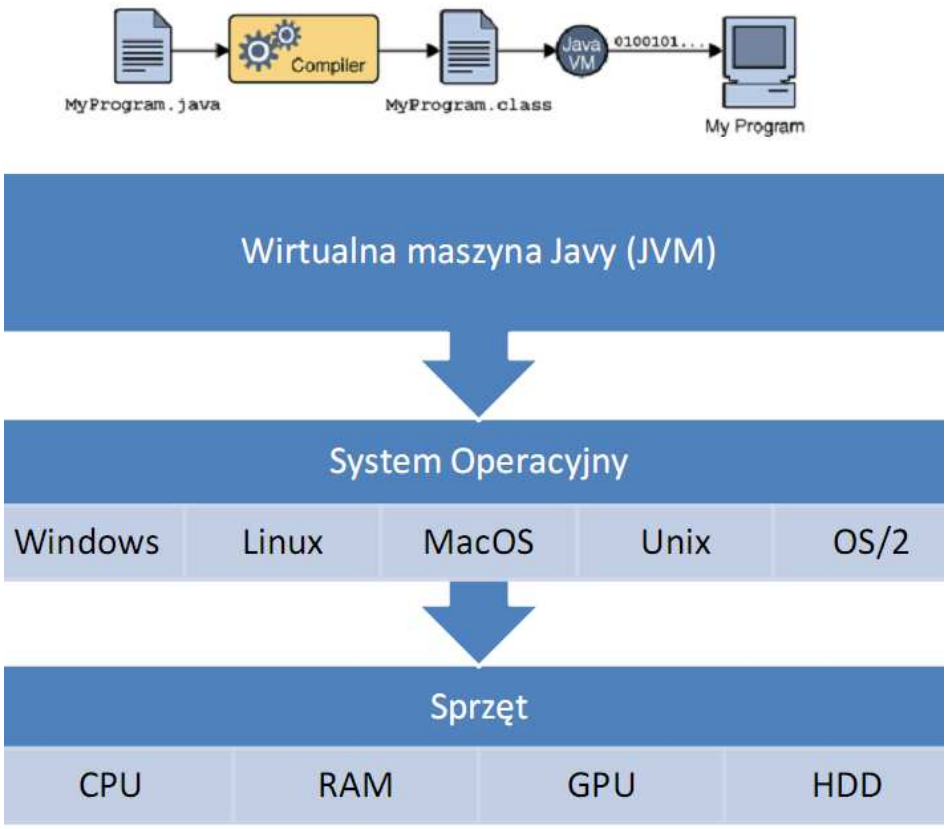
W podrozdziale tym zajmę się omówieniem języka Java. Dlaczego akurat wybrałem ten język, a nie np. technologię flash, która jest bardzo powszechna w aplikacjach internetowych? Powodów jest kilka. Dużą motywacją była dla mnie chęć podjęcia nauki tego języka oraz szerokie perspektywy na rynku pracy związane z jego znajomością. Innym ważnym powodem było stosunkowo łatwe przedstawienie i opisanie zagadnień fizycznych dzięki istnieniu licznych bibliotek i mechanizmów umożliwiających rysowanie elementów takich, jak koła, linie, prostokąty, itp..

Język Java jest obecnie bardzo popularnym językiem programowania i chyba nie ma na świecie programisty, który by o nim nie słyszał. Java jest znana nie tylko programistom, ale jest ona językiem powszechnie używanym w Internecie czy sieciach komputerowych. Wielu użytkowników Internetu nie jest nawet świadoma tego, ile gier, animacji, które przegląda każdego dnia, zostało stworzonych właśnie w tym języku. Stał się on szczególnie popularny w telefonach komórkowych. W ostatnich latach powstały setki, a nawet tysiące aplikacji dedykowanych na te urządzenia. Oprócz tak powszechnego zastosowania Javy do budowania małych i amatorskich apletów czy animacji, język ten stosuje się do pisania poważnych projektów, np. systemy kontroli w lotnictwie.

Język Java pojawił się pod koniec 1995 roku, jednak jego początki sięgają roku 1990. Wtedy to w raporcie „Further” Bill Boy zasugerował SUNowi stworzenie środowiska obiektowego na bazie C++ [4]. W bardzo krótkim czasie, bo już w roku 1991 powstał język OAK – „Object Application Kernel”. Został on stworzony przez zespół kierowany przez Jamesa Goslinga w firmie Sun Microsystems i był on przeznaczony dla aplikacji w elektronice powszechnego użytku (sterowanie pralkami, lodówkami, kuchenkami mikrofalowami, czyli wszystkim co zawiera mikroprocesor). Oficjalna nazwa języka Java pojawiła się dopiero w 1995 roku, kiedy to okazało się, że pierwotne określenie OAK jest nazwą już istniejącego języka programowania. Dynamiczny rozwój Javy sprawił, że w 1999 roku ukazało się jej nowe oblicze, Java 2.

Java jest językiem obiektowym, ale zawiera elementy programowania strukturalnego. Jego dużą zaletą jest zdarzeniowość i wielowątkowość. Standardowe i rozszerzone biblioteki tego języka zapewniają wszechstronność zastosowań. Oprócz

wcześniej wymienionych Java obejmuje takie zagadnienia, jak: karty inteligentne i elektronika, systemy zarządzania bazami danych, obsługa multimediów, Internet, grafika 3D, kryptografia, itd.. Jest środowiskiem bezpiecznym i zapewnia swoim aplikacjom niezwykłą kontrolę nad tym bezpieczeństwem. Niebywałym atutem Javy jest możliwość uruchamiania danego programu na różnym sprzęcie i różnych systemach operacyjnych. Kod wynikowy powstały w wyniku kompilacji stanowi tzw. beta-kod, który dopiero jest interpretowany przez Maszynę Wirtualną pracującą w danym środowisku. Tak więc Maszyna Wirtualna jest podstawą do uruchamiania programów napisanych w Javie. Dzięki temu pomysłowi możliwe stało się odseparowanie od pozostałych poziomów sprzętowo-programowych znajdujących się poniżej maszyny wirtualnej (Rys. 1.) [4].



Rys. 1. Technologia Javy. [6]

Oprócz Maszyny Wirtualnej JavaVM (Virtual Machine) drugim ważnym elementem są Java API (Application Programming Interfaces) – interfejsy tworzenia aplikacji. Oba elementy tworzą tzw. platformę Javy, która często nazywana jest środowiskiem uruchomieniowym aplikacji – Java Runtime Engine (JRE). Biblioteki klas, metod, pól, itp. (wymagane do interpretacji kodu) znajdują się w postaci skompilowanej

w Java API. Dla potrzeb tworzenia aplikacji SUN oferuje pakiet Java Development Kit (JDK lub Java SDK – Software Development Kit), który składa się z JRE oraz narzędzi kompilacji i bibliotek [4].

Podstawowymi zaletami, jakie niesie ze sobą używanie języka Java są:

- Niezależne od platformy środowisko programistyczne,
- Składnia języka podobna do składni C++,
- Język w pełni obiektowy,
- Programista nie ma bezpośredniej kontroli pamięci, usuwanie obiektów z pamięci odbywa się automatycznie w trzech etapach: odnajdywanie nieużywanych obiektów, kasowanie tych obiektów i defragmentacja pamięci. Zajmuje się tym Garbage Collector,
- Brak wskaźników (w zamian tablice), czyli zlikwidowanie kłopotu nadpisywania pamięci,
- Kontrola kodu (problem przypisania, a nie porównania w wyrażeniach warunkowych),
- Interfejsy na rzecz wielokrotnego dziedziczenia [5].

Dalszą część tego rozdziału poświęcę nowemu rozwiązaniu, jakie oferuje Java, czyli apletom. Ponieważ moja praca w głównej mierze dotyczy owych apletów, opiszę krótko czym one są, jakie są ich zalety i gdzie się ich używa.

Otóż aplet jest programem, który może być uruchamiany i kontrolowany przez inną aplikację, w kontekście której został uruchomiony. Najczęściej środowiskiem uruchomieniowym są przeglądarki WWW, np. Internet Explorer. Tak naprawdę aplet jest tworzoną przez programistę klasą, która dziedziczy z klasy JApplet (pakiet javax.swing.*), bądź też ze starszej biblioteki klasą Applet (pakiet java.awt.*).

Patrząc na niektóre kody źródłowe stworzonych przeze mnie apletów można dostrzec strukturę podobną do poniższej:

```
1. package paczka;
2. import javax.swing.JApplet;
3. public class NazwaKlasy extends JApplet{
4.
5.         @Override
```

```
6.      public void init() {
7.      }
8. }
```

Listing 1. Struktura apletów.

Używając takiej właśnie struktury programista tworzy aplet. Słowo „extends” i występująca po nim nazwa mówi, z jakiej klasy dziedziczy nowo tworzona klasa. Dziedziczenie mówi tyle, że nowo powstała klasa oprócz funkcji bezpośrednio w niej zdefiniowanych posiada również funkcjonalność klasy, z której dziedziczy. Dziedziczenie stosuje się często, kiedy chce się rozszerzyć możliwości jakieś klasy o nową funkcjonalność. Ja w swoich programach oprócz dziedziczenia z klasy „JApplet” zastosowałem dziedziczenie z klasy „Applet”. Nie wprowadzało to jednak istotnych zmian i równie dobrze mógłbym stosować tylko jedną z nich. Ważną częścią kodu jest część, w której importujemy biblioteki (linijka 2 kodu z listingu (Listing 1.)). Linijka „import java.applet.Applet;” informuje kompilator o korzystaniu z danej biblioteki. W podobny sposób importuje się pozostałe biblioteki użyte przez programistę w programie.

Aplety do sterowania swoim zachowanie wykorzystują pięć podstawowych metod:

- init(),
- start(),
- stop(),
- destroy(),
- paint(Graphics).

Pierwsza z nich „init()” jest metodą wywoływaną przez przeglądarkę jako pierwsza, przy ładowaniu strony, gdy inicjowany jest zawarty w niej aplet. Metodę tą wykorzystuje się do budowania graficznego interfejsu apletu, czyli do umieszczenia komponentów na aplecie.

Kolejna metoda „start()” jest wywoływana zaraz po „init()” oraz kiedy użytkownik wraca do wcześniej załadowanej strony. Stosuje się ją, kiedy ma się wykonać jakieś działanie niezależnie od działań użytkownika. Jest to metoda jednak rzadko stosowana. Metoda „stop()” jest metodą uruchamianą, gdy użytkownik opuszcza stronę WWW z danym apletem, a metoda „destroy()” wywoływana jest podczas zamykania apletu. Ostatnią z wymienionych metod jest metoda „paint(Graphics)”. Metoda ta jest wywoływana, kiedy zachodzi konieczność odmalowania powierzchni apletu lub poprzez jej wymuszenie

metodą „repaint()”. Jest to bardzo ważna metoda, gdyż powinno się w niej umieścić wszystkie polecenia rysujące na powierzchni apletu, aby nie zniknęły przy pierwszym jego odświeżeniu [7].

Powstaje jeszcze pytanie w jaki sposób umieścić aplet na stronie internetowej. Otóż robi się to umieszczając go w strukturze katalogów danej strony oraz dopisując kilka linijek w jej kodzie. Kod przykładowej strony, która jest tworzona automatycznie przez środowisko NetBeans (patrz rozdział) w trakcie debugowania (uruchamiania) apletu wygląda następująco:

```
<HTML>
<HEAD>
<TITLE>Applet HTML Page</TITLE>
</HEAD>
<BODY>

<H3><HR WIDTH="100%">Applet HTML Page<HR WIDTH="100%"></H3>

<P>
<APPLET codebase="classes" code="NewJApplet.class" width=350
height=200></APPLET>
</P>

<HR WIDTH="100%"><FONT SIZE=-1><I>Generated by NetBeans
IDE</I></FONT>
</BODY>
</HTML>
```

Listing 2. Struktura strony z apletem.

Jest jeszcze jedna i już ostatnia rzecz, na którą chciałbym zwrócić uwagę w tym rozdziale. Chodzi o wspomnianą wcześniej wielowątkowość, czyli możliwość wykonywania przez program różnych zadań w tym samym czasie. Język Java oferuje tworzenie tzw. wątków. Żeby było łatwiej zrozumieć ich działanie, posłużę się krótkim przykładem. Przypuśćmy, że programista będzie chciał stworzyć aplikację do przeglądania zdjęć. W aplikacji ma być wyświetlany bieżący czas. W takim wypadku programista może

umieścić kod obsługujący zegar (odliczający czas) w osobnym wątku, a na samym początku programu napisać linijkę kodu, która go uruchomi. Komputer nie czekając na wykonanie wszystkich instrukcji zawartych w danym wątku przejdzie do obsługi kolejnej linijki programu dając tym samym możliwość wykonywania innych zadań.

Często programiści cały swój program umieszczają w jednym głównym wątku. Jest to wątek event-dispatching (ED), który wykonuje „odmalowanie" kontrolek i obsługuje zdarzenia związane z kontrolkami. W wątku tym wykonywane są metody „paint()" i actionPerformed()". Wykonanie kodu w wątku ED możliwe jest dzięki klasie SwingUtilities udostępniającej dwie metody: „invokeLater()", która powoduje wykonywanie kodu w wątku ED i nie czeka na wykonanie wskazanego kodu, tylko kończy się zaraz po wywołaniu oraz „invokeAndWait()", która działa tak samo, jak poprzednia, z tym że czeka ona na zakończenie wskazanego kodu [7]. Jednej z nich użyłem w trzech moich apletach. W ciele metody „init()" każdego z tych apletów można dostrzec podobny zapis:

```
java.awt.EventQueue.invokeAndWait(new Runnable() {
        public void run() {
                //ciało metody init()
        }
});
```

Listing 3. Tworzenie wątku.

Odpowiada on za utworzenie wyżej wspomnianego wątku ED.

W moich apletach pojawia się również inny sposób tworzenia wątków. Do tego celu używa się klasy „Timer". Używając tej klasy można stworzyć harmonogram zadań, które mają zostać wykonane w przyszłości w jednym wątku działającym w tle. Zadania mogą być zaplanowane do pojedynczego wykonania lub powtarzane w regularnych odstępach.

```
Timer latanie = new Timer();
latanie.schedule(new TimerTask(){
        @Override
        public void run() {
                //instrukcje do wykonania
```

```
        }
    }, 0,50);
```

Listing 4. Tworzenie wątku z harmonogramem.

Nowe zadanie powstaje poprzez utworzenie obiektu klasy „TimerTask” oraz nadpisanie metody „run()” tej klasy w sposób widoczny powyżej (Listing 4.). Harmonogram wykonywania zadania ustala się poprzez wywołanie z odpowiednimi parametrami metody „schedule()” na obiekcie klasy „Timer”. Metoda „schedule()” przyjmuje różne zestawy parametrów. Ja we wszystkich apletach używam tego samego, czyli że pierwszym parametrem jest zadanie do wykonania, drugim opóźnienie wykonania i trzecim interwał czasowy, z jakim ma być powtarzane zadanie. Zadanie będzie wykonywane cyklicznie dopóki nie zostanie wywołana na nim metoda „cancel()”, którą można zauważyć w kodach źródłowych załączonych do pracy.

Myślę, że omówione przeze mnie do tej pory podstawowe funkcje języka Java w wystarczający sposób ułatwią zrozumienie działania apletów przedstawionych w kolejnych rozdziałach.

2.2. Netbeans

Środowiskiem programowania, jakie użyłem do tworzenia apletów w Javie jest środowisko Netbeans. Wybrałem to oprogramowanie, gdyż uważam, że nadaje się ono doskonale do programowania w tym języku. Dzięki bogatemu i czytelnemu interfejsowi użytkownika umożliwia szybkie i sprawne poruszanie się po jego zasobach. Dodatkową zaletą jest sprawdzanie poprawności pisanego kodu oraz wyświetlanie podpowiedzi w trakcie pisanie programu. Podpowiedzi są bardzo cenne iż wyświetlają sugestie poleceń, których nazwy zaczynają się od wprowadzanych przez programistę liter. Dzięki tym sugestiom nie tylko można szybko i zręcznie wprowadzać kolejne linijki kodu, ale także przesyłać prawidłowe parametry do zdefiniowanych metod. Do danej sugestii dołączony jest również jej opis mówiący za co dany element (klasa, metoda, właściwość itp.) jest odpowiedzialny i jakie parametry przyjmuje. Opisy te w dużej mierze pomogły w scharakteryzowaniu obiektów używanych podczas tworzenia apletów i stanowiły podstawę do powstania kolejnych rozdziałów, w których opisuję kody źródłowe tychże programów. Środowisko NetBeans umożliwia tworzenie gotowych aplikacji okienkowych lub też samych klas. Graficzny interfejs środowiska pozwala na łatwe dodawanie kontrolek

do powierzchni tworzonej aplikacji poprzez przeciąganie ich z palety gotowych elementów. Korzystając z menu właściwości danej kontrolki w prosty sposób można ustawić podstawowe jej właściwości oraz związane z nią zdarzenia.

Ogólnie rzecz biorąc „NetBeans IDE jest środowiskiem programistycznym (Integrated Development Environment - IDE) - narzędziem służącym do tworzenia, kompilowania, uruchamiania i testowania programów. Całe IDE jest napisane w Javie, jednak umożliwia ono również tworzenie programów w innych językach. Ponadto (oprócz NetBeans Mobility Pack i NetBeans Profiler) dostępna jest duża ilość modułów rozszerzających jego możliwości. NetBeans IDE jest produktem dostępnym za darmo oraz bez żadnych ograniczeń co do jego używania” [8] i tym samym jest to dodatkowa i ważna zaleta tego środowiska.

2.3. Microsoft Office Publisher 2003

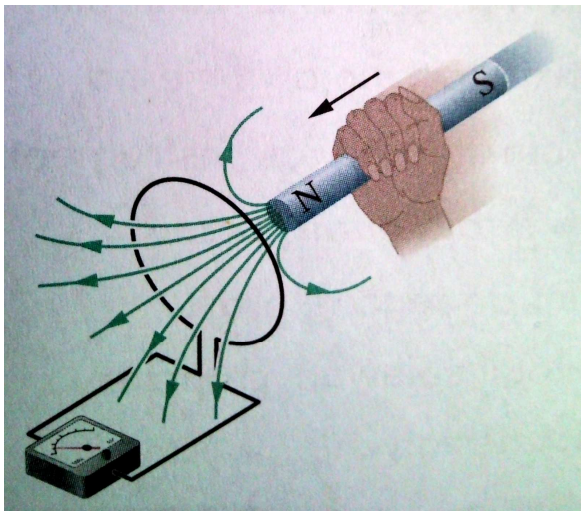
We wcześniejszych rozdziałach opisałem język i środowisko, w którym tworzyłem aplety. Aby aplety tworzyły jedną, spójną logiczną całość, utworzyłem witrynę internetową i umieściłem je w niej wraz ze stosownymi opisami. Do stworzenia witryny posłużyłem się gotowym szablonem oferowanym przez program Microsoft Office Publisher 2003. Dzięki temu programowi mogłem również poumieszczać aplety na konkretnych stronach, dodawać dodatkowe elementy (np. logo uczelni) oraz w łatwy sposób dopisać tekst i odpowiednio go sformatować. Dzięki menu tworzonemu automatycznie na panelu bocznym podczas dodawania do projektu nowych stron użytkownik ma możliwość dotarcia z każdej strony do dowolnego miejsca w witrynie. Niewątpliwą zaletą, którą daje Publisher jest możliwość publikacji witryny.

Program Publisher daje o wiele więcej możliwości niż te opisane przeze mnie powyżej. Jednak na potrzeby mojego projektu były one w zupełności wystarczające. Zainteresowanych tym programem oraz jego możliwościami odsyłam na stronę Microsoftu, gdzie sam producent opisuje program w bardziej szczegółowy sposób.

3. Indukcja elektromagnetyczna i prezentacja tego zjawiska za pomocą apletów

3.1. Indukcja elektromagnetyczna – definicja

Zjawisko indukcji elektromagnetycznej objaśnię na przykładzie zaczerpniętym z książki Resnick'a, Halliday'a i Walker'a „Podstawy Fizyki 3". Jest tam opisane doświadczenie z przewodzącą pętlą i wsuwanym do (wysuwanym z) niej magnesem sztabkowym (Rys. 2.).



Rys. 2. Gdy magnes wchodzi do (wychodzi z) pętli to pojawia się w niej płynący prąd wskazywany za pomocą podłączonego miernika [1].

Na powyższym rysunku (Rys. 2.) pętla połączona jest z bardzo czułym miernikiem prądu, który wykrywa nawet najmniejsze prądy pojawiające się w obwodzie. Ponieważ do obwodu nie jest podłączone jakiekolwiek zasilanie, prąd w nim nie płynie. Doświadczenie polega na tym, aby poruszać magnesem sztabkowym wsuwając go do pętli i wysuwając. Gdy magnes jest wsuwany na mierniku można zaobserwować wychylenie się wskazówki, co oznacza pojawienie się prądu w obwodzie. Gdy jednak magnes zatrzymamy, to nie zaobserwujemy wychylenia wskazówki, a prąd w obwodzie nie będzie płynął. Podobnie, jak przy wsuwaniu magnesu dzieje się podczas jego wysuwania, z tym że wskazówka miernika wychyli się w drugą stronę [1].

Wykonując powyższe czynności kilka razy nasuwa się kilka wniosków:

- Gdy magnes porusza się względem pętli, to pojawia się prąd, a gdy nie ma względnego ruchu pętli i magnesu przepływu prądu się nie zaobserwuje.
- Przybliżanie i oddalanie północnego bieguna magnesu do pętli powoduje przepływ prądu w dwie różne strony. Gdy biegun zmieniony zostanie na południowy, też przepłynie prąd z tym, że w przeciwnym kierunku niż było to w przypadku bieguna północnego.
- Przez obwód płynie prąd o większym natężeniu, gdy magnes poruszany jest szybciej [1].

Ponieważ prąd płynący w pętli nie powstaje w wyniku dołączenia do obwodu zewnętrznego źródła zasilania, ale powstaje w wyniku zmiany pola magnetycznego pochodzącego od magnesu, prąd ten nazywany jest prądem indukowanym. Do wytworzenia prądu potrzebna jest praca (przypadająca na jednostkę ładunku). Taka praca nazywana jest indukowaną siłą elektromotoryczną (SEM), a zjawisko wytwarzania prądu i SEM nazywa się zjawiskiem indukcji elektromagnetycznej.

Powyżej wspomniane jest o zmianie pola magnetycznego. Równoważne jest to ze zmianą liczby linii pola magnetycznego. Pozwala to nam na sformułowanie prawa indukcji Faradaya: „SEM jest indukowana w pętli, gdy zmienia się liczba linii pola magnetycznego, przechodzących przez pętlę”.

Aby określić wartość SEM, konieczne będzie najpierw określenie „ilości pola magnetycznego” przechodzącego przez pętlę. W tym celu zdefiniuję strumień magnetyczny. Gdy pętla obejmująca pewną powierzchnię S zostanie umieszczona w polu magnetycznym o indukcji $\vec{B}$, to strumień magnetyczny będzie opisany następującym wzorem:

$$\phi_B = \int \vec{B} \cdot d\vec{S} \quad (1)$$

Strumień magnetyczny przechodzący przez powierzchnię S.

Powyższy wzór jest określony dla powierzchni o dowolnym kształcie oraz zarówno do pól jednorodnych, jak nie niejednorodnych. Można jednak uprościć ten wzór dla szczególnego przypadku, gdzie pole jest jednorodne, a jego linie są prostopadłe do płaszczyzny objętej przez to pole. Wtedy iloczyn skalarny we wzorze (1) można zapisać jako $\vec{B} \cdot d\vec{S} = B \cdot dS \cdot \cos 0° = B \cdot dS$, wartość B wynieść przed znak całki, a pozostałe

wyrażenie $\int dS$ jest po prostu polem powierzchni objętym przez pętlę. Dzięki powyższym zabiegom wzór (1) sprowadza się do postaci:

$$\phi_B = B \cdot S \qquad (2)$$

$\vec{B} \perp S$, $\vec{B}$ jednorodne.

Jednostką strumienia magnetycznego w układzie SI jest tesla razy metr kwadratowy i nosi ona nazwę webera (Wb).

Dzięki zdefiniowaniu strumienia magnetycznego można teraz prawo Faradaya przedstawić w bardziej użyteczny sposób: „Wartość SEM indukowanej w przewodzącej pętli jest równa szybkości, z jaką strumień magnetyczny , przechodzący przez tę pętlę zmienia się w czasie” (3).

$$\varepsilon = -\frac{d\phi_B}{dt} \qquad (3)$$

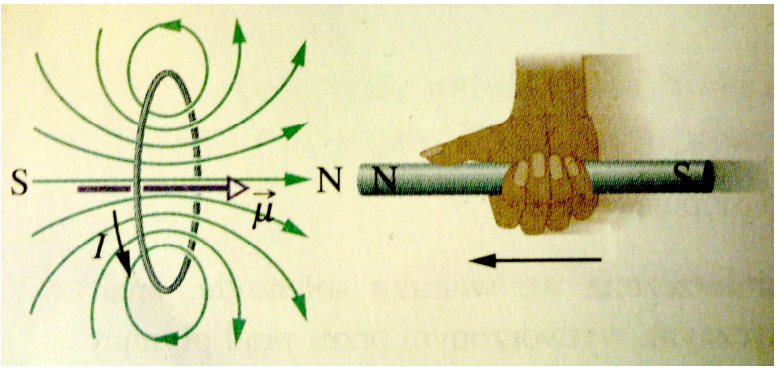
Prawo Faradaya.

Często mamy do czynienia nie z pojedynczą pętlą, ale z kilkoma zwojami, czyli tzw. cewką. Gdy strumień magnetyczny będzie przenikał wszystkie zwoje cewki, to w każdym z tych zwojów pojawi się indukowana SEM, a całkowita wartość SEM będzie równa sumie tych parcjalnych SEM. Wtedy wzór (3) przyjmie postać ze wzoru (4), gdzie N będzie oznaczało ilość zwojów.

$$\varepsilon = -N\frac{d\phi_B}{dt} \qquad (4)$$

Prawo Faradaya dla cewki o N zwojach.

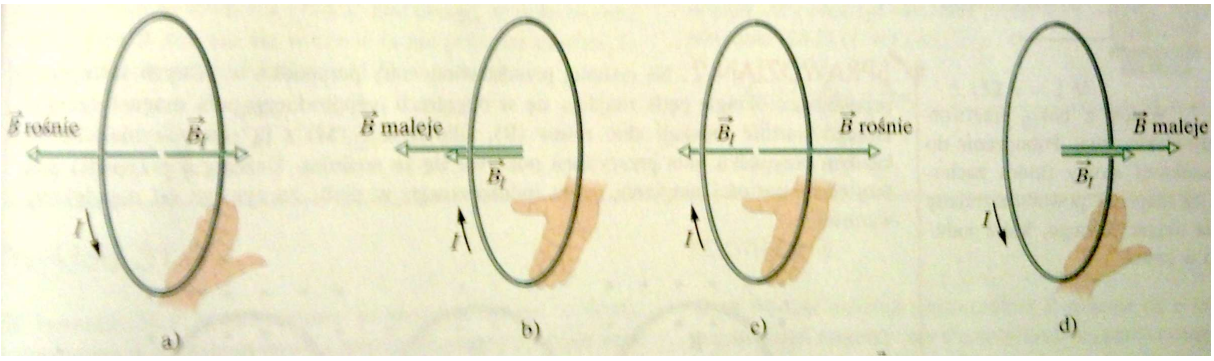
We wzorze (3) i (4) pojawia się minus. Jego obecność oznacza, że indukowana SEM ε przeciwdziała zmianie strumienia. Jest to ważne, gdy chcemy wyznaczyć kierunek indukowanego w obwodzie prądu. Tą prawidłowość zawdzięczamy Heinrichowi Friedrichowi Lenzowi, który niedługo po odkryciu prawa indukcji przez Faradaya, sformułował regułę zwaną dziś regułą Lenza: „Prąd indukowany płynie w takim kierunku, że pole magnetyczne wytworzone przez ten prąd przeciwdziała zmianie strumienia pola magnetycznego, które ten prąd indukuje”.



Rys. 3. Magnes przesuwany w kierunku pętli.

Zbliżając magnes tak, jak na rysunku (Rys. 3.) strumień przenikający przez pętlę zwiększa swoją wartość (linie pole skierowane są w lewo). W celu zapobiegania wzrostowi tego strumienia indukowany prąd I musi wytworzyć swoje własne pole, skierowane wewnątrz pętli w prawą stronę. Tym samym nowo powstałe pole przeciwdziała temu wytwarzanemu przez magnes i zgodnie z regułą prawej dłoni prąd musi płynąć przeciwnie do ruchu wskazówek zegara (patrząc od strony magnesu).

Ponieważ nowe pole przeciwdziała zmianom strumienia magnetycznego, a zmiany te mogą być dodatnie i ujemne bez względu na to, w którą stronę skierowane jest pole magnesu, to nie zawsze nowo powstałe pole jest przeciwnie skierowane do pola magnesu. Przedstawiają to rysunki (Rys. 4.).



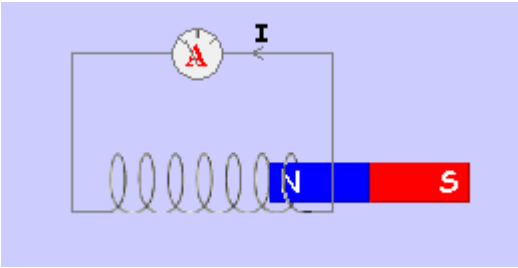
Rys. 4. Pętle z prądem.

Wektor indukcji $\vec{B}_I$ (nowo powstałe pole) jest zawsze skierowany przeciwnie do wzrastającego wektora indukcji pola $\vec{B}$, natomiast jest zgodny z kierunkiem malejącego

wektora indukcji pola $\vec{B}$. Reguła prawej dłoni wskazuje kierunek prądu indukowanego, w zależności od kierunku indukowanego pola [1].

3.2. Aplety komputerowe jako sposób przedstawiania zjawiska indukcji elektromagnetycznej.

W części praktycznej swojej pracy, czyli stronie internetowej z apletami zamieściłem sześć apletów przedstawiających różne zagadnienia z indukcji elektromagnetycznej oraz jeden związany z elektrostatyką. Pierwsze cztery aplety pokazują sposoby wzbudzania prądu indukcyjnego, o którym mówiłem już w poprzednim rozdziale. Wsuwanie magnesu do zwojnicy i wysuwanie jest analogicznym przypadkiem do tego z rysunku (Rys. 2.), z tą różnicą, że na rysunku jest pojedyncza pętla (zwój), a w aplecie (Rys. 5.) jest zwojnica składająca się z kilku zwojów.

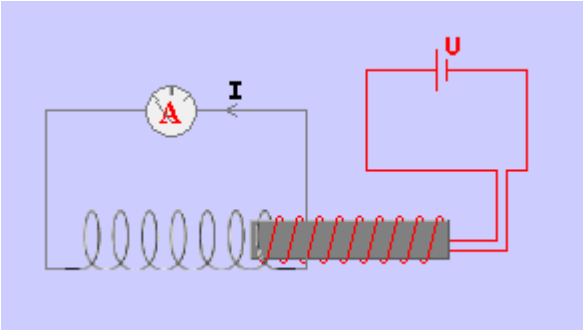


Rys. 5. Wsuwanie i wysuwanie magnesu (ruch magnesu względem zwojnicy)

Chcąc obliczyć SEM indukowaną w obwodzie skorzystamy więc ze wzoru (4), a nie z (3) stosowanego dla pojedynczej pętli. Kierunek płynącego prądu jest wyznaczany za pomocą reguły prawej dłoni, czyli analogicznie, jak dla przypadku z rysunku (Rys. 3.).

Na rysunku (Rys. 5.) widnieje sytuacja, kiedy magnes jest wsuwany do zwojnicy i odpowiada ona rysunkowi (Rys. 4a.).

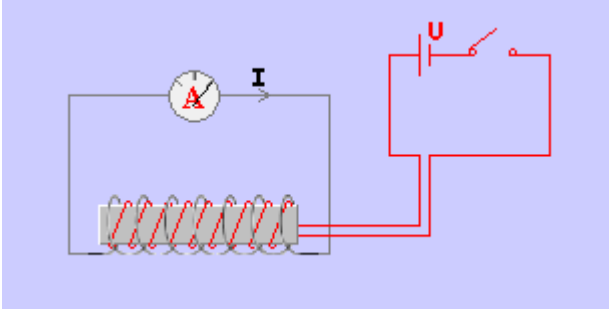
Kolejny aplet jest bardzo podobny, z tym że zamiast zwykłego magnesu sztabkowego zastosowano elektromagnes.



Rys. 6. Wsuwanie (wysuwanie) elektromagnesu do (z) zwojnicy.

Mimo tej różnicy idea nie różni się niczym od pierwszego apletu. Na rysunku (Rys. 6.) nie widać dokładnie, jak zwoje elektromagnesu są podłączone do zasilania. Sytuacja przedstawiona jest tu w taki sposób, jakby biegun północny elektromagnesu znajdował się po jego lewej stronie, a południowy po prawej.

Na trzecim aplecie (Rys. 7.) widzimy, że elektromagnes umieszczony jest na stałe w zwojnicy. Jedyną czynność, która jest wykonywana przez operatora jest włączanie i wyłączanie prądu w zwojnicy elektromagnesu.

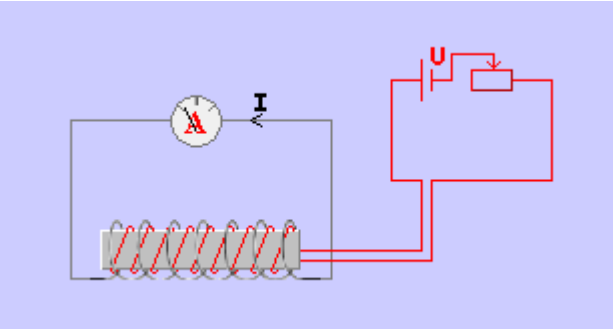


Rys. 7. Włączanie i wyłączanie prądu w zwojnicy.

Podczas włączenia następuje gwałtowny wzrost płynącego prądu od zera do wartości zasilania. W chwili włączenia widać momentalne wychylenie wskazówki w lewą stronę. Podczas wyłączenia zasilania zaobserwujemy podobny efekt, ale tym razem wskazówka wychyli się na chwilę w prawo. Gdy przełącznik jest w ustalonej pozycji amperomierz nie wskazuje przepływu prądu, gdyż w obwodzie nic się nie indukuje. Spowodowane jest to tym, że pole magnetyczne nie zmienia się w tym czasie. Wykorzystując wiedzę z poprzedniego rozdziału, a dokładnie prawo Faradaya (4) widzimy, że do powstania prądu indukcyjnego w zwojnicy potrzebny jest zmienny strumień magnetyczny przechodzący przez jej powierzchnię. Tutaj strumień jest stały.

Jedyne zmiany strumienia powodowane są przełączaniem przełącznika, kiedy to prąd w obwodzie elektromagnesu gwałtownie zmienia swoją wartość.

Czwarty aplet (Rys. 8.) wygląda podobnie, jak trzeci (Rys. 7.). Jedyną różnicą jest zastąpienie przełącznika potencjometrem. Zmiana strumienia magnetycznego przechodzącego przez powierzchnię zwojnicy uzyskiwana jest tutaj poprzez ciągłą zmianę oporu na potencjometrze.



Rys. 8. Zmiana natężenia prądu w uzwojeniu elektromagnesu przy pomocy opornicy suwakowej.

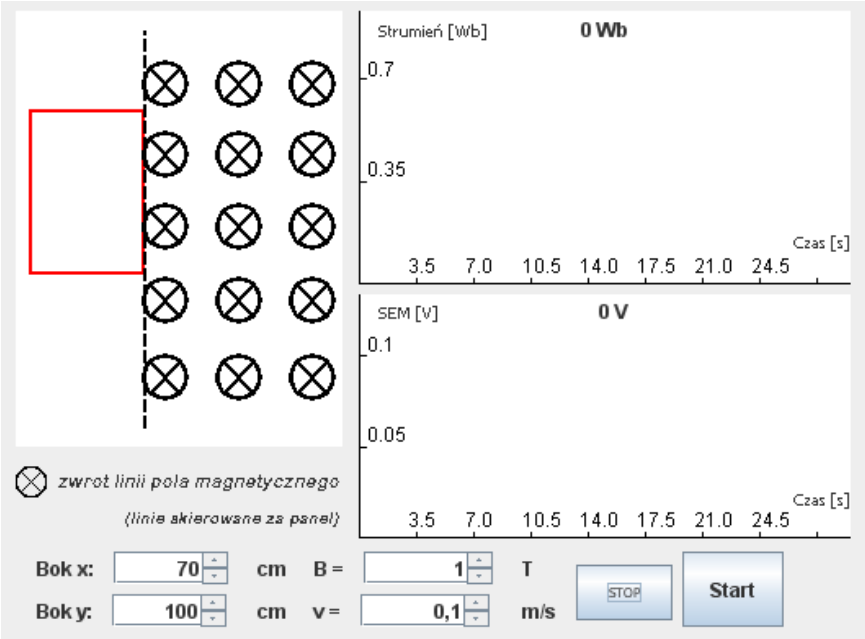
Zmiana oporu zgodnie z definicją oporu (5) powoduje zmianę prądu w obwodzie elektromagnesu, co z kolei wpływa na powstawanie zmiennego pola magnetycznego. Gdy opór na potencjometrze rośnie, to przy stałym napięciu zasilania U, płynący w obwodzie prąd maleje.

$$R = \frac{U}{I} \qquad (5)$$

Definicja oporu. R – opór, U – napięcie, I – prąd. [1]

Gdy zmiana strumienia magnetycznego jest dodatnia tzn. rośnie wartość indukcji magnetycznej pola wytwarzanego przez elektromagnes, w zwojnicy indukuje się prąd i wskazówka amperomierza wychyla się w lewo. Natomiast, gdy wartość indukcji maleje, wskazówka wychyla się w prawo.

Teraz omówię bardziej interaktywne aplety, w których użytkownik ma możliwość ustawiania różnych parametrów sterujących zachowaniem apletu. Zacznę od apletu, który obrazuje wchodzenie przewodzącej ramki w pole magnetyczne (Rys. 9.).



Rys. 9. Ramka wchodząca w pole magnetyczne.

Na rysunku (Rys. 9.) widzimy ramkę, która może się poruszać w prawo (w rzeczywistym doświadczeniu ramka może być poruszana ręcznie przez człowieka lub przez maszynę). Obok ramki i rysunku przedstawiającego pole magnetyczne znajdują się wykresy. Można na nich zaobserwować, jak zmienia się w czasie strumień magnetyczny przechodzący przez powierzchnię objętą ramką oraz SEM indukowaną w obwodzie podczas wchodzenia ramki w pole. Użytkownik w każdej chwili może rozpocząć poruszanie ramki, wstrzymać lub też przenieść ją do pozycji początkowej, gdzie ramka znajduje się tuż przed polem magnetycznym. Na wykresach umieszczone są również kontrolki, które pozwalają odczytać bieżącą wartość strumienia lub SEM. Użytkownik dzięki interaktywności apletu może również za pomocą odpowiednich kontrolek ustalać wymiary ramki (x,y), wartość indukcji pola magnetycznego(B) oraz prędkości (v), z jaką ramka ma wchodzić w to pole.

Chcąc narysować wykres strumienia magnetycznego musiałem skorzystać ze wzoru (1). Jednakże ze względu na to, że w naszym przypadku pole jest jednorodne i wektor indukcji $\vec{B}$ jest prostopadły do powierzchni ramki skorzystałem z prostszego wzoru $\phi_B = B \cdot S$ (2). Obszar ramki objęty przez pole magnetyczne zwiększa się wraz z przesuwaniem się ramki. Dlatego wzór (2) trzeba tak zmodyfikować, aby zawierał te wielkości, które mamy dane. Pole powierzchni ramki jest prostokątem, więc można do wzoru (2) za S podstawić wzór (6).

$$S = x \cdot y \ \ [m \cdot m] \qquad (6)$$

Pole powierzchni prostokąta.

Naszym zmieniającym się parametrem będzie długość ramki x. Skorzystamy tutaj z przekształcenia wzoru (7) na drogę w ruchu jednostajnym, gdyż takim porusza się nasza ramka.

$$x = v \cdot t \ \ [\frac{m}{s} \cdot s] \qquad (7)$$

Droga w ruchu jednostajnym, v – prędkość, t – czas ruchu

Po wszystkich tych przekształceniach wzór (2) przyjmuje postać wzoru (8).

$$\phi_B = B \cdot y \cdot v \cdot t \qquad (8)$$

Wzór na strumień magnetyczny.

Korzystając ze wzoru (8) można już obliczyć wartość strumienia magnetycznego przechodzącego przez powierzchnię ramki objętą tym strumieniem w danej chwili t. Wartość strumienia osiąga maksymalną wartość, kiedy cała ramka wejdzie w pole magnetyczne, czyli w chwili $t = \frac{x}{v}$. Wtedy strumień przestaje rosnąć liniowo i ma swoją ustaloną wartość.

Spójrzmy teraz, jak zachowuje się wykres SEM indukowanej w ramce. Widać, że podczas wchodzenia ramki w pole utrzymuje się on na stałej określonej wysokości. Gdy ramka już w całości znajdzie się w polu wartość SEM jest równa zeru. Aby potwierdzić te spostrzeżenia obliczeniami trzeba skorzystać z prawa Faradaya $\varepsilon = -\frac{d\phi_B}{dt}$ (3). We wzorze tym widzimy, że indukowana SEM jest pochodną strumienia po czasie. Podstawmy więc wzór (8) do wzoru (3).

$$\varepsilon = -\frac{d}{dt}\left(B \cdot y \cdot v \cdot t\right) \qquad (9)$$

SEM

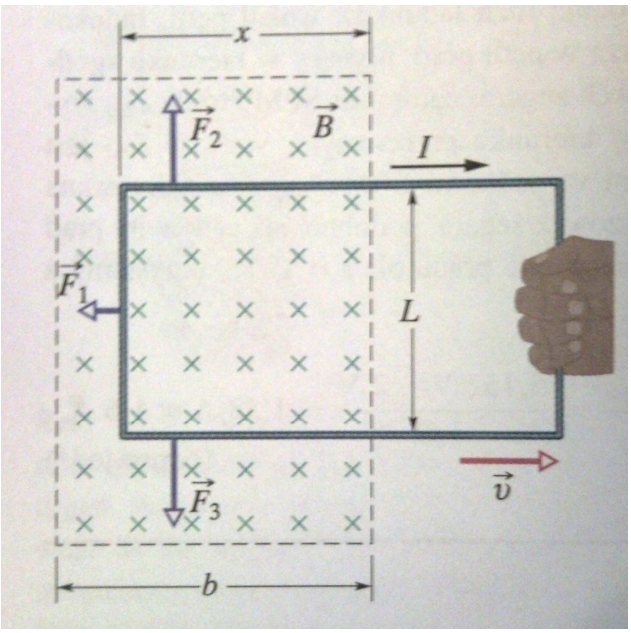
Po zróżniczkowaniu wzór (9) przyjmuje postać znacznie prostszego wzoru (10).

$$\varepsilon = -(B \cdot y \cdot v) \qquad (10)$$

SEM

Korzystając z ostatniego wzoru można obliczyć wartość indukowanej SEM. Dla dowolnej chwili wartość SEM jest stała i równa wartości obliczonej tym wzorem. SEM jest pochodną strumienia po czasie, a strumień po wejściu całej ramki w pole przyjmuje stałą wartość, więc SEM od tego momentu jest równa zeru (pochodna ze stałej równa jest zeru).

Patrząc na rysunek (Rys. 9.) można by się zastanowić, jakie siły działają na ramkę, która jest wpychana w pole magnetyczne. Rozkład tych sił przedstawia rysunek (Rys. 10.).



Rys. 10. Ramka wchodząca w pole i rozkład sił.

Na rysunku (Rys. 10.) kierunek indukowanego prądu oznaczono zgodnie z regułą prawej dłoni, gdzie kciuk wskazuje kierunek wektora indukcji $\vec{B}$, a pozostałe (zakrzywione wzdłuż obwodu ramki) palce kierunek prądu. Weźmy pod uwagę trzy odcinki ramki, które podczas jej wchodzenia w pole będę się w nim znajdowały. Będą na nie działały prostopadłe siły oznaczone na rysunku $\vec{F}_1$, $\vec{F}_2$ i $\vec{F}_3$. Siły $\vec{F}_1$ i $\vec{F}_2$ ze względu na symetrię mają jednakowe wartości i są przeciwnie skierowane. Dzięki temu wzajemnie się znoszą. Trzecia siła $\vec{F}_1$ równa jest co do wartości sile z jaką ramka jest wpychana

w pole, lecz przeciwnie skierowaną (zapobiegająca wpychaniu). Jej wartość można obliczyć ze wzoru na siłę działająca na prostoliniowy przewodnik z prądem (11).

$$\vec{F}_B = I \cdot \vec{L} \times \vec{B} \qquad (11)$$

Siła działająca na przewodnik z prądem.

$\vec{L}$ jest wektorem długości skierowanym zgodnie z kierunkiem przepływem prądu przez przewodnik. Niewiadomą we wzorze (11) będzie I, czyli wartość indukowanego prądu. Wartość tą można jednak obliczyć korzystając ze wzoru (12).

$$I = \frac{\varepsilon}{R} \qquad (12)$$

Indukowany prąd. ε - SEM, R - opór przewodnika

Wystarczy znać opór przewodnika, z którego wykonana jest ramka oraz wyznaczoną wartość SEM. Wartość prądu możemy zapisać wzorem (13) wstawiając do wzoru (12) wzór (10).

$$I = \frac{B \cdot y \cdot v}{R} \qquad (13)$$

Indukowany prąd.

Kiedy już znamy wartość prądu można ze wzoru (11) wyznaczyć wartości sił. W przypadku odcinka, na który działa siła $\vec{F}_1$ wektor długości $\vec{L}$ jest prostopadły do wektora $\vec{B}$ i wynosi $90°$. Dzięki temu możemy wyznaczyć wartość siły $\vec{F}_1$ korzystając ze wzoru na iloczyn wektorowy i znacznie uprościć obliczenia (14).

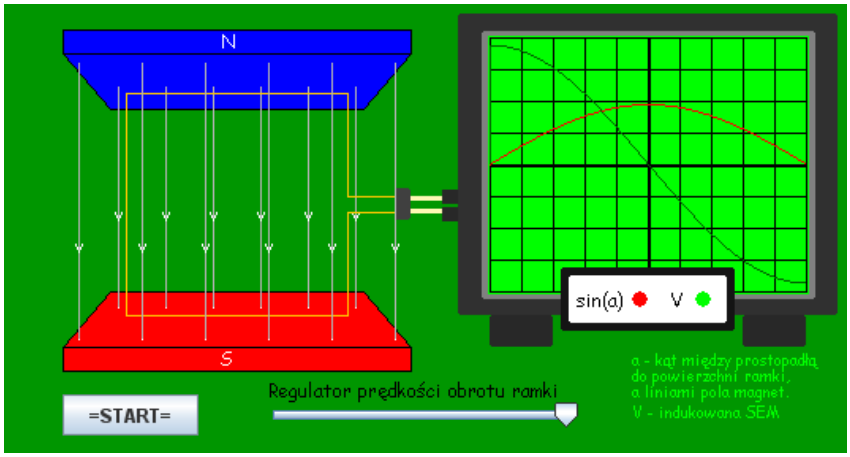
$$F_1 = I \cdot L \cdot B \cdot \sin 90° = I \cdot L \cdot B = I \cdot y \cdot B \qquad (14)$$

Podstawiając równanie (13) do równania (14) otrzymujemy ostateczny wzór na siłę F_1 (15).

$$F_1 = \frac{B^2 \cdot y^2 \cdot v}{R} \qquad (15)$$

To już koniec rozważań nad ramką wchodzącą w pole. Wnioskiem z tego doświadczenia jest fakt, że pole magnetyczne próbuje wypchnąć ramkę, kiedy człowiek chce ją do niego wepchnąć. Ponieważ wektory sił leżą w płaszczyźnie ramki, to nie próbują obracać ramką. Zaobserwujemy to przy omawianiu kolejnego apletu, kiedy ramka będzie się obracać, a jej płaszczyzna nie będzie prostopadła do linii pola magnetycznego. Podsumowując praca, którą wykonuje człowiek wpychając ramkę w pole zostaje całkowicie przekształcona w energię termiczną w ramce.

Ostatnim już apletem, przedstawiającym jeden ze sposobów indukowanie prądu, jest aplet z obracającą się przewodzącą ramką w jednorodnym polu magnetycznym (Rys. 11.). Jest to uproszczony mechanizm prądnicy.



Rys. 11. Obracająca się ramka w jednorodnym polu magnetycznym.

Użytkownik może za pomocą przycisku „START” rozpocząć obracanie ramki lub wstrzymać jej obrót. Za pomocą regulatora może również modyfikować prędkość obrotu (przesuwając wskaźnik). O obserwacjach napiszę trochę później. Najpierw skupię się na fizycznym rozwiązaniu problemu. Na oscyloskopie obok rysowane są dwa wykresy. Jeden jest sinusem kąta pomiędzy normalną (styczną) do powierzchni ramki, a liniami pola magnetycznego biegnącymi od bieguna północnego do bieguna południowego. Wykres sinusa jest po prostu liczony na podstawie odchylenia normalnej od linii pola. Inaczej i bardziej skomplikowanie ma się sprawa z SEM indukowaną w ramce. Tutaj trzeba skorzystać ze wzorów fizycznych. Obliczając wartość SEM można postąpić analogicznie

do przykładu poprzedniego, kiedy ramka była wpychana w pole. Tutaj ramka jest już cała umieszczona w polu, a zmiana powierzchni, przez którą przenika strumień magnetyczny jest uzyskiwana poprzez obrót ramki. Skorzystajmy więc z wzoru (1) ($\phi_B = \int \vec{B} \cdot d\vec{S}$). Podobnie, jak to robiliśmy wcześniej, można uprościć ten wzór, gdyż pole jest jednorodne. Wtedy iloczyn skalarny we wzorze (1) można zapisać jako $\vec{B} \cdot d\vec{S} = B \cdot dS \cdot \cos\alpha$, wartość B wynieść przed znak całki, a pozostałe wyrażenie $\int dS$ jest po prostu polem powierzchni objętym przez ramkę. Dzięki powyższym zabiegom wzór (1) sprowadza się do postaci:

$$\phi_B = B \cdot S \cdot \cos\alpha \qquad (16)$$

$\vec{B}$ jednorodne, α – kąt między normalną do powierzchni ramki, a liniami pola magnetycznego, S – powierzchnia ramki.

Ponieważ w czasie obrotu ramki kąt α zmienia się w czasie i jest zależny od prędkości obracania się ramki jego wartość określona jest wzorem (17).

$$\alpha = \omega \cdot t \qquad (17)$$

ω – prędkość obrotu (kątowa), t – czas obrotu.

Zauważmy, że ramka startuje z pewnej początkowej pozycji, gdzie kąt początkowy pomiędzy normalną, a liniami pola magnetycznego jest już równy $90°$. Uwzględniając to powyższy wzór przyjmie następującą postać:

$$\alpha = 90° + \omega \cdot t \qquad (18)$$

Podstawiając wzór (18) do wzoru (16) otrzymujemy gotowy wzór na strumień magnetyczny:

$$\phi_B = B \cdot S \cdot \cos(90° + \omega \cdot t) \qquad (19)$$

Teraz zajmijmy się obliczeniem SEM indukowanej w obwodzie. Aby to zrobić wstawmy równanie (19) do równania (3) ($\varepsilon = -\frac{d\phi_B}{dt}$). Otrzymamy:

$$\varepsilon = -\frac{d\left(B \cdot S \cdot \cos\left(90° + \omega \cdot t\right)\right)}{dt} \qquad (20)$$

Po zróżniczkowaniu:

$$\varepsilon = B \cdot S \cdot \omega \cdot \cos\left(-\omega \cdot t\right) \qquad (21)$$

SEM w obracającej się ramce.

Wartość prądu w ramce wyznaczmy ze wzoru (12) podstawiając do niego równanie (21). Otrzymamy:

$$I = \frac{B \cdot S \cdot \omega \cdot \cos\left(-\omega \cdot t\right)}{R} \qquad (22)$$

Prąd indukowany w ramce.

Dla przedstawienia danego problemu nie jest konieczne przedstawienie rozkładu sił działających na ramkę. Powiem tylko, że siły te dążą do umieszczenia ramki w poziomie, a dokładnie do tego, aby linie pola magnetycznego były prostopadłe do powierzchni ramki. Siły te są równoważone przez siłę, którą działa człowiek na ramkę obracając ją.

Wykonując to doświadczenie widać, że największa (co do wartości bezwzględnej) wartość SEM indukuje się wówczas, gdy linie pola magnetycznego padają prostopadle do powierzchni ramki, a równa się zeru, kiedy padają równolegle. Zauważyłem również, że gdy ramka obraca się szybciej, tym większa SEM się w niej indukuje. Zależność tą widać we wzorze (21), gdzie prędkość kątowa ω jest składnikiem mnożącym wartość amplitudy SEM.

4. Definicja elektrostatyki i jej krótka prezentacja za pomocą apletu.

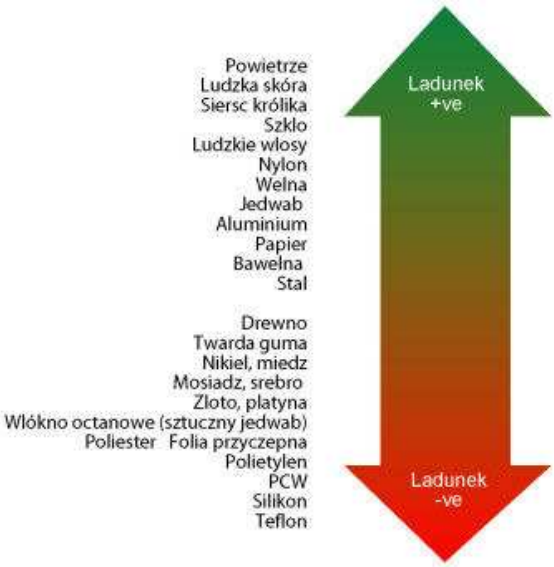
4.1. Elektrostatyka – definicja

Elektrostatyka jest to dział fizyki zajmujący się zjawiskami wzajemnego oddziaływania pomiędzy nieruchomymi ładunkami elektrycznymi [2].

Aby w pełni wyjaśnić to zagadnie konieczne będzie poznanie kilku pojęć. Jednym z nich jest ładunek elektryczny. Każde ciało składa się z cząstek elementarnych takich, jak

protony, neutrony i elektrony. Protony i elektrony posiadają taką właściwość, jak ładunek elektryczny. Pierwsze naładowane są dodatnio (posiadają dodatni ładunek elektryczny), a drugie z nich naładowane są ujemnie (posiadają ujemny ładunek elektryczny). Ciała, które mają takie same ilości ładunków dodatnich i ujemnych nazywane są ciałami elektrycznie obojętnymi (ciałami o zerowym ładunku wypadkowym). Jeżeli przeważa w ciele ilość ładunków dodatnich, to ciało jest naładowane dodatnio, a jak ujemnych, to ujemnie. Każde ciało posiada ogromne ilości ładunku elektrycznego, a przejawy jego występowanie w świecie można obserwować w życiu codziennym. Patrząc się na niebo w czasie burzy możemy dostrzec błyskawicę. Kiedy zdejmujemy sweter czasem słyszymy przeskakujące iskry. Rodzice często pokazywali nam, jak po potarciu plastikowej linijki szmatką przyciągała ona do siebie drobne papierki [1].

Sposobów elektryzowania ciał, czyli przekazywania im ładunku (przekazywania elektronów) jest kilka i są nimi: elektryzowanie przez tarcie, dotyk i indukcję [2]. Ponieważ moja praca w głównej mierze dotyczy indukcji elektromagnetycznej, nie będę się więcej zajmował elektrostatyką. W kolejnym rozdziale omówię tylko jeden aplet, który prezentuje sposób elektryzowania plastikowej linijki przez pocieranie futrem. Poniżej zamieszczam jeszcze rozkład materiałów pod względem biegunowości i wielkości gromadzonego przez nie ładunku podczas tarcia (Rys. 12.). Materiały po stronie dodatniej szeregu będą wykazywały tendencję do oddawania swoich elektronów powierzchniowych i naładują się dodatnio, podczas gdy materiały po stronie ujemnej będą dążyły do pozyskiwania elektronów, a więc do uzyskania ładunku ujemnego [3].

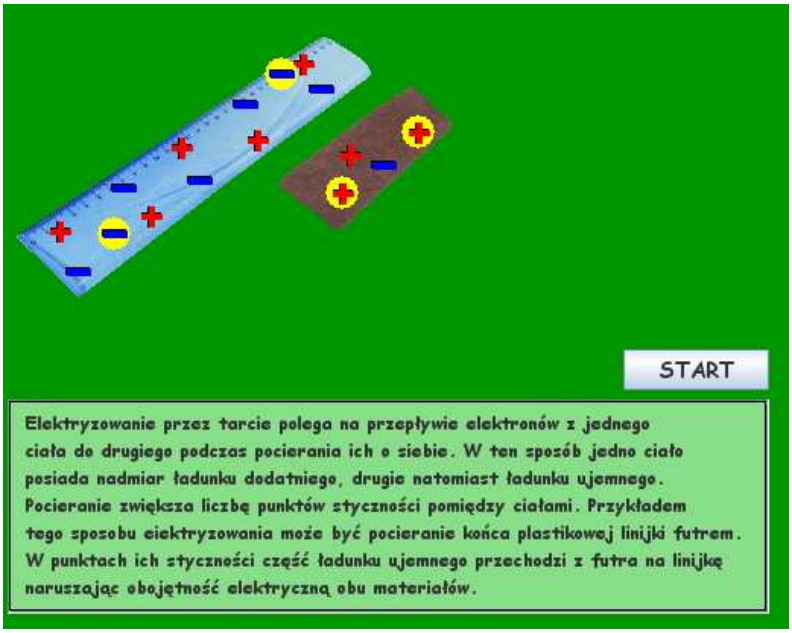


Rys. 12. Szereg tryboelektryczny [3].

4.2. Elektryzowanie przez tarcie – aplet

Na wstępie tego podrozdziału powiem czym jest elektryzowanie przez tarcie. Otóż polega ono na przepływie elektronów z jednego ciała do drugiego podczas pocierania ich o siebie, dzięki czemu jedno posiada nadmiar ładunku dodatniego, a drugie ujemnego [2]. Materiały, z których wykonane są ciała wykazują różną tendencję do oddawania swoich elektronów powierzchniowych. Te po stronie dodatniej szeregu tryboelektrycznego (Rys. 12.) będą chciały oddać elektrony i naładują się dodatnio, podczas gdy materiały po stronie ujemnej będą dążyły do pozyskiwania elektronów, a więc do uzyskania ładunku ujemnego. Im silniejsze będzie tarcie, tym więcej będzie punktów styczności pomiędzy ciałami i tym samym elektryzowanie będzie silniejsze [3].

Zjawisko elektryzowania przez tarcie przedstawiłem za pomocą apletu (Rys. 13.).



Rys. 13. Elektryzowanie przez tarcie – aplet.

Widnieje na nim plastikowa linijka i szmatka (futerko). Użytkownik wciskając przycisk „START” może rozpocząć pocieranie linijki szmatką. Podczas pocierania zauważyć można przechodzenie elektronów ze szmatki na linijkę. Nadmiarowe elektrony, które pojawiły się na linijce oraz nadmiarowe protony pozostałe na szmatce podświetlone są na żółto. W wyniku pocierania oba materiały przestały być elektrycznie obojętne, linijka została naładowana ujemnie, a szmatka dodatnio.

5. Aplety komputerowe.

5.1. Wsuwanie i wysuwanie magnesu.

Aplet ten został utworzony jako klasa rozszerzająca jedną ze standardowych klas języka Java, a mianowicie klasę JApplet. Z właściwościami tej klasy można było zapoznać się w rozdziale o języku Java i środowisku, w którym były tworzone aplety, gdyż tam opisuję podstawowe metody języka Java oraz ich przeciążanie, jakie stosuję w swoich apletach.

W kodzie źródłowym omawianego apletu można dopatrzyć się sześciu zdefiniowanych metod, z czego dokładnie trzy (init(), paint(Graphics g), update(Graphics g)) są metodami przeciążonymi. Ściśle związaną z nimi metodą jest użyta metoda paintComponet(Graphics g). Metody te są często stosowane w apletach, także nie będę się w tym rozdziale rozpisywać o ich własnościach. Każdy kto chciałby dowiedzieć się o nich więcej powinien zajrzeć do rozdziału, który polecałem wyżej mówiąc o klasie JApplet.

Jednakże dla każdego apletu ciało takich przeciążonych metod może wyglądać nieco inaczej i zawierać np. wywołania innych metod zdefiniowanych przez użytkownika w programie albo po prostu wykonywać konkretne metody zdefiniowane w środowisku, jak metodę „drawImage()”, która odpowiada za rysowanie obrazka na danym obiekcie, na którym została wywołana. Nie wystarczy więc przedstawić ogólnego zarysu metod, jakie opisane są w „polecanym” przeze mnie rozdziale, ale dla każdego apletu należy zrobić to osobno.

Zanim jednak przejdę do omawiania poszczególnych metod, chciałbym przedstawić zmienne, których używam w programie. Deklarację zmiennych umieściłem na początku głównej klasy (zazwyczaj tak się robi). Niektóre z nich zainicjowałem określonymi wartościami potrzebnymi w programie. Przed nazwami zmiennych znajduję się nazwy określające typ danej zmiennej. W przypadku zmiennych „bufferGraphics” i „offscreen” przed nazwą typu umieszczony jest modyfikator dostępu „protected”. Jeśli przed typem nie ma umieszczonego modyfikatora, to standardowo jest to zmienna prywatna (z modyfikatorem „private”).

Modyfikatory dostępu, mają następujące znaczenie:

- public - wszystkie klasy mają dostęp do pól danych i metod public,

- private - dostęp do metod i pól danych posiadają jedynie inne metody tej samej klasy,
- protected - metoda lub pole danych protected lub może być używana jedynie przez metody swojej klasy oraz metody wszystkich jej klas pochodnych,
- package - jest to modyfikator domyślny, wszystkie metody i pola danych bez modyfikatora dostępu traktowane są jako typu package. Metody (lub pola danych) typu package mogą być używane przez inne klasy danego pakietu [9].

Teraz zajmijmy się metodami użytymi w programie. Zacznijmy od ciała metody „init()” (Listing 5.), gdyż jak przypominam jest to pierwsza metoda, która wykonuje się po uruchomieniu apletu.

```
41  public void init() {
42      setSize(300,150);
43      JPanel pp = new Jpanel();
44      pp.setPreferredSize(new Dimension(300,150));
45      pp.setBackground(new Color(204,204,255));
46      add(pp);
47      try
48      {
49          img = getImage(getDocumentBase(), „spirala.gif”);
50          img1 = getImage(getDocumentBase(), „spirala1.gif”);
51      }catch(Exception ex)
52      {
53          System.out.println(„Nie udalo się zaladowac obrazków”);
54      }
55      Uruchom();
56  }
```

Listing 5. Metoda init().

W pierwszej linijce kodu tej metody (wiersz 42) ustawiany jest rozmiar apletu. Odpowiada za to metoda setSize(300,150), która jako argumenty przyjmuje kolejno szerokość i wysokość wyrażone przez liczby typu int. Oczywiście wartości te odpowiadają liczbom pikseli. W kolejnym wierszu (wiersz 43) tworzony jest panel (obiekt klasy

Jpanel), po którym będziemy rysować. Natomiast w dwóch kolejnych ustawiane są takie parametry, jak rozmiar panelu, za który odpowiada metoda „setPreferredSize()” wywołana na panelu z argumentem, który jest obiektem klasy „Dimension”, a który zawiera konkretne wymiary, kolejno szerokość i wysokość.

Metoda „setBackground()” ustawia tło panelu. Tutaj parametrem jest kolor, który jest obiektem klasy „Color”. Konstruktor klasy „Color” przyjmuje tutaj trzy argumenty typu int, którą są wartościami składowych koloru RGB.

Po ustawieniu właściwości panelu trzeba jeszcze go dodać do powierzchni apletu za pomocą metody add(), która jako parametr przyjmuje dany obiekt.

Instrukcje 49 i 50 służą utworzeniu obiektów typu Image z przechowywanych na dysku gotowych obrazków. Można by oczywiście ich nie tworzyć, tylko odczytywać obrazy z dysku za każdym razem, kiedy zajdzie potrzeba ich użycia. Ja jednak tego nie robię, gdyż operacje dyskowe są kosztowne i wpłynęłoby to niekorzystnie na wydajność apletu. Za tworzenie obiektów „Image” z obrazów znajdujących się na dysku odpowiedzialna jest metoda „getImage()”, która przyjmuje dwa parametry. Pierwszym jest ścieżka do pliku na dysku, a drugim nazwa obrazka. Tutaj zamiast typowej ścieżki widzimy metodę „getDocumentBase()”, która zwraca ścieżkę (URL) do dokumentu, w którym osadzony jest aplet i oznacza to, że obrazki związane z apletem powinny być umieszczone w katalogu, w którym znajduje się dokument (tu: strona html), w którym jest osadzony aplet. Czasem bywa jednak tak, że obrazki z jakichś powodów mogą nie znajdować się w wybranej lokalizacji (się. mogę zostać przypadkowo usunięte z katalogu bądź przeniesione). Wówczas aplet nie będzie miał możliwości ich załadowania i spowoduje to wygenerowanie wyjątku. W takim przypadku program działałby nieprawidłowo lub mógłby w ogóle się nie uruchomić. W celu uniknięcia takiej sytuacji instrukcje 49 i 50 zostały zawarte w bloku „try-catch”, a dokładniej w sekcji „try” tego bloku, natomiast sekcja „catch” służy do obsługi powstałego wyjątku i najczęściej zawarta jest w niej informacja o rodzaju błędu. Tak jest i tutaj. W przypadku nie załadowania obrazka program zwróci informację „Nie udalo się zaladowac obrazków”.

Po poprawnym wykonaniu bloku „try-catch” program przechodzi do obsługi metody Uruchom() i wykonania instrukcji w niej zawartych. Na tym kończy się działanie przeciążonej metody init().

Metodę Uruchom() omówię pod koniec tego rozdziału. A teraz zapoznajmy się z tym, co dzieje się w metodzie paint() (Listing 6.), która jako parametr przyjmuje obiekt typu „Graphics”.

```
66    public void paint(Graphics g) {
67        if (bufferGraphics == null)
68        {
69            offscreen = createImage(getWidth(), getHeight());
70            bufferGraphics = offscreen.getGraphics();
71        }
72         paintComponent(bufferGraphics);
73         g.drawImage(offscreen, 0, 0, null);
74    }
```

Listing 6. Metoda paint().

Metoda „paint()” jest metodą klasy „Component” i jest wywoływana, kiedy konieczne jest odmalowanie komponentu apletu, np. podczas pierwszego uruchomienia apletu czy po powrocie na stronę apletu.

Do programowego wywołania metody „paint()” służy metoda „repaint()”. Jednak nie wywołuje ona tej metody bezpośrednio tylko odwołuje się do metody „update()”, która zawiera w swoim ciele wywołanie metody paint().

Patrząc na powyższy kod metody „paint()” osoby, które mają już jakieś doświadczenie z językiem Java z łatwością zauważą, że pełni ona rolę podwójnego buforowania.

Otóż co to takiego i po co stosuje się podwójne buforowanie?

Aby samemu zauważyć, jakie zalety niesie podwójne buforowanie wystarczy w ciele metody „paint()” zamiast obecnej zawartości użyć wywołania metody „paintComponent(g)”. Wtedy kod metody „paint()” wyglądałby następująco:

```
public void paint(Graphics g)
{
        paintComponent(g);
}
```

Listing 7. Metoda paint().

Zauważylibyśmy powstawanie smug podczas odświeżania obrazu ruchomego. Dzieje się tak, ponieważ obraz jest odświeżany bezpośrednio na obiekcie graficznym

i wtedy „widzimy” proces jego odmalowywania. To jest tak, jakbyśmy widzieli nasz portret malowany przez artystę. Nie jest tak, że siadamy na krzesełko i … nagle przed nami stoi nasz gotowy portret. Obserwujemy każdy ruch malarza, każde pociągnięcie jego pędzla. Natomiast inaczej jest, kiedy zamówimy jakiś obraz. Nie widzimy wtedy procesu jego malowania, ale końcowy rezultat pracy malarza, kiedy odbieramy jego dzieło. I podobnie jest z podwójnym buforowaniem. Tworzy ono najpierw pusty obraz o rozmiarach naszego apletu, tzw. off-screen, a z niego nowy kontekst graficzny, maluje na nim i dopiero później zwraca nam gotowy obraz do naszego pierwotnego kontekstu graficznego, jakim jest powierzchnia apletu. Za tworzenie pustego obrazu odpowiada polecenie „offscreen = createImage(getWidth(), getHeight())”, gdzie „offscreen” jest obiektem zadeklarowanym na początku klasy naszego apletu jako zmienna „protected Image” (chroniona, patrz.). Metoda „createImage()” tworzy obiekt typu „Image” o zadanych (jako parametry) rozmiarach. Tutaj parametry pobierane są za pomocą metod „getWidth()” i getHeight()”, które zwracają odpowiednio szerokość i wysokość komponentu. W tym przypadku komponentem jest sam aplet. Następnie z obiektu „offscreen” za pomocą metody „getGraphics()” tworzony jest nowy kontekst graficzny „bufferGraphics” zadeklarowany na początku klasy apletu jako „protected Graphics”.

Tworzenie obiektów „offscreen” i bufferGraphics wykonuje się tylko raz w trakcie działania apletu („if (bufferGraphics == null)”) i można byłoby je zamieść w metodzie „init()”. Jednak ja wolałem umieścić je w metodzie „paint()”, aby było widać, że są od z nią silnie związane i stanowią nieodłączną całość z ciałem tej metody.

W kolejnej linijce kodu wywoływana jest metoda „paintComponent()”, do której jako parametr podawany jest nowy kontekst graficzny, na którym odbywa się odmalowywanie komponentów. Gotowy obraz w postaci obiektu „offscreen”, na którym był utworzony kontekst graficzny „bufferGraphics” zostaje w całości odmalowany, na właściwym, pierwotnym kontekście graficznym „g”, a efekty obserwujemy na ekranie. Dzięki takiemu buforowaniu widzimy ostry obraz i płynnie poruszający się magnes.

Dodatkowym ulepszeniem może być i jest użyte w kodzie tego apletu, przeciążenie metody „update()”. Oryginalna, nieprzeciążona metoda „update()” najpierw czyści ekran wypełniając cały obszar apletu kolorem tła, następnie odtwarza oryginalne kolory i dopiero wywołuje metodę „paint()”. W naszym przypadku pierwsze dwie czynności są zbędne i czasami mogą niekorzystnie wpłynąć na jakość ruchomych obrazów. Po przeciążeniu metoda ta przyjmuje następująca postać:

```
public void update(Graphics g) {
    paint(g);
}
```

Listing 8. Metoda update().

W tym miejscu kończę opisywanie metod wywoływanych automatycznie i sposobu ich przeciążania. Teraz zajmę się opisem tych metod, które odpowiadają za tworzenie konkretnych obrazów i kształtów wyświetlanych na ekranie (magnesu, zwojnicy, obwodu, itd.). W pewnym momencie w ciele metody „paint()” wywoływana jest zdefiniowana przeze mnie metoda „paintComponent()” z parametrem „Graphics” (Listing 9.).

```
public void paintComponent(Graphics g){
    Graphics2D g2=(Graphics2D)g;
    super.paint(g2);
    rysuj(g2);
}
```

Listing 9. Metoda paintComponent().

W pierwszym wierszu w ciele tej metody rzutuję dany kontekst graficzny (tutaj będzie nim utworzony bufferGraphics, który jest przekazywany przez parametr) na nowy obiekt typu „Graphics2D”. Klasa „Graphics2D” jest rozszerzeniem klasy „Graphics” o dodatkową funkcjonalność, taką jak: zaawansowana kontrola geometrii, transformacja położenia, zarządzanie kolorami czy rozmieszczenie tekstu. Jest to podstawowa klasa do renderowania (sposobu wyświetlania) dwuwymiarowych kształtów, tekstu i obrazów.
Dzięki powyższemu rzutowaniu otrzymuję nowe możliwości tworzenia naszej grafiki. W dalszej kolejności wywołuję metodę „super.paint()”, do której przekazywany jest nasz nowy, rozszerzony kontekst „g2”. Wywołanie powyższej metody różni się od zwykłego wywołania „paint()” tym, że w tym przypadku wywoływana jest bazowa, a nie przeciążona metoda „paint()”. W końcu wywołuję metodę „rysuj()” z parametrem w postaci obiektu „g2” i tym samym przechodzę do właściwej metody rysującej konkretne kształty i obrazy.

Omawianie tej metody zacznę może od użytych w niej metodach wywoływanych na przekazanym do niej przez parametr kontekście graficznym. Dokładniej rzecz biorąc opowiem o metodach, jakie wykorzystuje klasa „Graphics2D”, gdyż nasz kontekst jest właśnie tego typu. Jedną z takim metod jest metoda „drawImage()”. Do metody tej

możemy przekazywać różne zestawy parametrów. Ja jednak ze względu na ich mnogość ograniczę się do użytych w programie, a są to kolejno: obiekt typu „Image”, współrzędne x i y (typu int) położenia na danym kontekście graficznym oraz obiekt typu „ImageObserver”. Jednym słowem „drawImage()” maluje obraz na danym kontekście graficznym zaczynając od jego górnego-lewego rogu. Ostatni wymieniony parametr określa obiekt, który jest informowany, kiedy obraz jest przetworzony i najczęściej jest to „this”, czyli bieżący obiekt.

Jeśli obraz jest już w całości załadowany i jego piksele nie są przetwarzane to metoda „drawImage()” zwraca zawsze „true”. W każdym innym wypadku zwraca wartość „false” i informuje o tym „ImageObserver’a”.

Wywołując metodę „setColor()” na kontekście graficznym ustawia się kolor, w jakim będziemy rysować po danym kontekście. Jako parametr można użyć standardowej biblioteki kolorów np. „Color.blue”, czyli kolor niebieski. Zamiast „blue” można wpisać inne nazwy kolorów. Można również zdefiniować swój własny kolor na podstawie palety kolorów RGB i wtedy wygląda to następująco: „new Color(r,g,b)”, gdzie „r”, „g” i „b” są odpowiednimi liczbami typu „int” od 0 do 255, które określają nasycenie koloru czerwonego, zielonego i niebieskiego.

Inną użyteczną metodą jest metoda rysująca prostokąt. Jednak musimy wiedzieć czy chcemy rysować obiekt wypełniony kolorem czy też tylko jego kontur. Jeżeli ma być wypełniony kolorem to stosujemy metodę „fillRect()”, a jeżeli nie to „drawRect()”. Istnieje również inna metoda, która dodaje takiemu prostokątowi efekt trójwymiarowości i jest nią „fill3DRect()” (bądź „draw3DRect()”). Nie będę omawiać osobno tych czterech metod, gdyż są one bardzo podobne, tylko skupię się na metodzie „fill3DRect()”, którą stosuję w obecnym aplecie. Metoda ta przyjmuje pięć parametrów. Dwa pierwsze odpowiadają za położenie obiektu (typ „int”), dwa kolejne za jego szerokość i wysokość (typ „int”), a ostatni parametr (typu boolean) odpowiada za trójwymiarowość i określa czy obiekt ma wystawać ponad powierzchnię (wartość „true”) czy też ma być w niej zatopiony (wartość „false”).

Oprócz rysowania obrazów i prostokątów bardzo pomocna byłaby funkcja, która pozwalałaby rysować napisy. I tu z pomocą przychodzi metoda „drawChars()”, która rysuje znaki pobierane z ówcześnie zdefiniowanej przez nas tablicy znaków (char [] etykieta = {'A','I','S','N'}). Przyjmuje ona pięć parametrów. Pierwszym jest oczywiście tablica znaków. Kolejny („offset”) określa położenie znaku w tablicy, od którego chcemy

zacząć wypisywanie. Trzeci parametr odpowiada za ilość kolejnych znaków. Dwa ostatnie określają położenie napisu na kontekście graficznym.

Oczywiście każdy napis musi mieć swój format. Użytkownicy rozmaitych edytorów tekstu wiedzą, że można ustawić między innymi rodzaj, styl i wielkość czcionki, którą się pisze. I tak jest w przypadku wypisywania znaków na powierzchni kontekstu graficznego. Czcionka ustawiana jest przy użyciu metody „setFont()”, której parametrem jest konkretna czcionka. Aby wyjaśnić, jak ustawia się poszczególne parametry czcionki posłużę się przykładem „new Font("Comic Sans MS",1,14)”. Przedrostek „new” oznacza utworzenie nowego obiektu typu „Font”, czyli w tym przypadku nowej czcionki. Konstruktor klasy „Font” przyjmuje trzy parametry. Kolejno: nazwę czcionki (np. „Comic Sans MS”), styl (zwykła, pogrubiona, kursywa czy pogrubiona i kursywa) oraz wielkość. Nazwa czcionki jest standardową nazwą czcionki, które są powszechnie używane. Styl czcionki określony jest przez liczbę typu „int” będącą maską bitową danego stylu. Jeśli maska nie odpowiada jednej z oczekiwanych określających styl, to ustawiany jest styl „zwykły”.

Kolejną ciekawą metodą do rysowania figur jest metoda „drawPolyline()”, która rysuje sekwencję połączonych linii zdefiniowanych przez tablice koordynatów „x” i „y”. Stąd też dwoma pierwszymi argumentami są zdefiniowane uprzednio tablice (int[] x, int[] y). Każda para (x,y) definiuje punkt. Jeśli ostatni punkt jest różny od pierwszego to figura jest nie zamknięta. Za pomocą trzeciego parametru podaje się liczbę punktów.

W omawianym aplecie oprócz kanciastych figur istnieją również figury owalne, jak na przykład koło będące tarczą miernika. Do tego celu posłużyłem się metodą „drawOval()” (oraz „fillOval()”), która służy do rysowania tego typu obiektów. Jej pierwsze dwa parametry decydują o położeniu, a dwa kolejne odpowiednio o szerokości i wysokości figury.

To już była ostatnia metoda, która została użyta w ciele funkcji „rysuj()”. Poniżej zamieszczam treść funkcji rysującej (Listing 10.).

```
85  public void rysuj(Graphics2D g){
86          g.drawImage(img1, 60, 90, this);
87          g.setColor(Color.blue);
88          g.fill3DRect(165-i, 95, 50, 20, true);
89          g.setColor(Color.red);
90          g.fill3DRect(215-i, 95, 50, 20, true);
```

```
91
92      g.setColor(Color.white);
93      g.setFont(new Font("Comic Sans MS",1,14));
94      g.drawChars(etykieta, 3,1,170-i, 110);
95      g.drawChars(etykieta, 2,1,250-i, 110);
96
97      g.drawImage(img, 60, 90, this);
98      g.setColor(Color.gray);
99      g.drawPolyline(xPoints, yPoints, 6);
100
101     //tarcza
102     g.setColor(new Color(240,240,240));
103     g.fillOval(100,28, 25, 25);
104     g.setColor(new Color(120,120,120));
105     g.drawOval(100,28, 25, 25);
106     g.fill3DRect(112, 29, 2, 4, true);
107     g.drawLine(103, 32, 106, 35);
108     g.drawLine(122, 32, 119, 35);
109
110     g.setColor(Color.red);
111     g.setFont(new Font("Times New Roman",1,17));
112     g.drawChars(etykieta, 0,1,107,47);
113
114     g.setColor(Color.black);
115     g.setFont(new Font("Comic Sans MS",1,12));
116     g.drawChars(etykieta, 1,1,142,35);
117
118     g.setColor(new Color(120,120,120));
119     if (a==0)
120             g.drawLine(112,45,106+dyg,35-dyg); //wskazówka lewo
121     else
122             g.drawLine(112,45,120-dyg,35+dyg); //wskazówka prawo
123     xStrzalka[1]=145+zwrot;
124     xStrzalka[2]=145+zwrot;
```

```
125        g.drawPolyline(xStrzalka, yStrzalka, 4); //rysowanie strzałki
126     }
```

Listing 10. Metoda rysuj().

Omówiłem już funkcjonalność większości z używanych metod, dlatego też kod metody „rysuj()” omówię grupami poleceń, a nie wiersz po wierszu.

Polecenia 86-90:

Rysują w odpowiednich kolorach magnes na powierzchni apletu. Ponieważ magnes jest ruchomy, to dodana jest zmienna „i”, która zmienia jego położenie w poziomie.

Polecenia 92-95:

Podpisują odpowiednie bieguny magnesu.

Polecenia 97-99:

Rysują obwód ze zwojnicą.

Polecenia 102-108:

Odpowiadają za stworzenie tarczy amperomierza.

Polecenia 110-116:

Tworzą etykietę na tarczy amperomierza i etykietę oznaczającą prąd.

Polecenia 118-122:

Rysują odpowiednio skierowaną wskazówkę w zależności od kierunku ruchu magnesu. Nie ma to bezpośrednio sprawdzania ustawienia biegunów, co jak wiemy ma przede wszystkim znaczący wpływ na kierunek wskazówki, gdyż ich położenie jest z góry ustalone. Za sprawdzenie, w którą stronę porusza się magnes odpowiada zmienna „a” umieszczona w warunku funkcji „if-else”.

Polecenia 123-125:

Rysują strzałkę zwrotu prądu odpowiednio skierowaną w zależności od kierunku ruchu magnesu. Za kierunek odpowiada tu wartość zmiennej „zwrot”, która modyfikuje odpowiednie współrzędne „x” strzałki.

Ostatnią metodę, którą zajmę się w tym rozdziale nazwałem „Uruchom()”. Nazwa ta pojawiła się już przy omawianiu metody „init()”, która wywołuje tą metodę. Definicja „Uruchom()” wygląda następująco:

```
129 public void Uruchom()
130 {
131     Timer latanie = new Timer();
132     latanie.schedule(new TimerTask(){
133
134         @Override
135         public void run() {
136             if(i<a)
137             {
138                 a=50;
139                 i++;
140                 zwrot=5;
141             }
142             else
143             {
144                 a=0;
145                 i--;
146                 zwrot=-5;
147             }
148             dyg=-dyg;
149             repaint();
150         }
151     }, 0,40);
152 }
```

Listing 11. Metoda Uruchom(). Tworząca nowy wątek.

Może zacznę od powiedzenia czym zajmuje się ta metoda. Otóż odpowiada ona za poruszanie magnesu poprzez kontrolę i modyfikację odpowiadających za to zmiennych

oraz za wywoływanie funkcji „repaint()”, o której wspomniałem przy omawianiu przeciążonej metody „paint()”.

W powyższym kodzie pojawiają się nieznane nam dotąd elementy. Oczywiście nie są one obce czytelnikowi, który wcześniej zapoznał się z treścią rozdziału, w którym opowiadam o języku Java i o jego funkcjonalności, bądź osobom, które nie są nowicjuszami w programowaniu w Javie.

Pierwsze polecenie tworzy obiekt „latanie” klasy Timer. Klasa ta służy do tworzenia nowych wątków. Na temat samych wątków nie będę się rozpisywać, a osoby zainteresowane odsyłam do rozdziału o języku Java. Wróćmy do naszej klasy. Dzięki klasie „Timer()” możemy planować wykonywanie zadań w tle. To znaczy po uruchomieniu wątku program oprócz tego, że wykonuje polecenia określone w ciele danego zadania, to wykonuje dalej kod programu, który jest umieszczony za wywołaniem procedury uruchamiającej wątek. Zadania mogą być planowane tak by wykonać się tylko raz albo wykonywać się cyklicznie w regularnych interwałach czasowych.

Zaplanowanie konkretnego zadania odbywa się poprzez wywołanie metody „schedule()” na obiekcie klasy „Timer()”, czyli w naszym przypadku na obiekcie „latanie” (patrz: wiersz 132 w kodzie programu). Metoda „schedule()” przyjmuje trzy parametry. Pierwszym jest nowe zadanie „new TimerTask()”, a tuż za nim umieszczona w nawiasach klamrowych jego treść. Kolejnym jest obiekt typu „Date”, który decyduje o tym, kiedy zostanie uruchomione wykonywanie zadania. O klasie „Date” nie będę pisał, ponieważ chcę by zadanie było wykonywane natychmiast i w miejsc tego argumentu wpisuję „0”. Trzeci argument określa w milisekundach interwał czasu, co jaki będzie powtarzane wykonywanie zadania.

Zostało mi jeszcze do objaśnienia to, co dzieje się w ciele naszego zadania. Otóż, nadpisana (przeciążona) jest tam metoda run() (wiersz 135), w której ciele umieszczamy polecenia do wykonania przez to zadanie. Wiersze 136-147 wykonują funkcję warunkową „if-else”. Aby zrozumieć, co sprawdzane jest w warunku muszę powiedzieć do czego służą zmienne w nim umieszczone. Zmienna „a” jest to zakres poruszania się magnesu, a zmienna „i” (na początku równa „0”) służy do poruszenia magnesem (jest częścią składowej x położenia magnesu) i jest inkrementowana (zwiększa wartość o 1) dopóki nie osiągnie wartości „a=50”, potem jest dekrementowana (zmniejsza wartość o 1) póki nie osiągnie wartości „a=0”. Proces powtarza się cyklicznie. Widzimy, że zmienia jest również znak zmiennej „zwrot”. Patrząc do metody „rysuj()” można już łatwo dostrzec, jak te zmiany zmiennych poruszają magnesem.

W wierszu 148 zmienia się również znak zmiennej „dyg”, która jest zainicjowana wartością „1”. Zmienna „dyg” wywołuje efekt drgania wskazów modyfikując jej odpowiednie parametry określające położenie.

Na końcu (wiersz 149) wywoływana jest metoda „repaint()” (patrz strona), która powoduje (nie bezpośrednio) uruchomienie metody „rysuj()”, a tym samym odrysowanie naszego apletu z nowymi parametrami.

Kończąc ten rozdział chcę podkreślić jego ogromną wartość dydaktyczną, gdyż szczegółowy opisu tworzenia apletu pozwala stwierdzić, że rozdział ten można traktować jako element podstawowego kursu programowania w języku Java.

5.2. Wsuwanie i wysuwanie elektromagnesu.

Aplet ten od strony technicznej, jak i fizycznej bardzo podobny jest do apletu poprzedniego, czyli „wsuwania i wysuwania magnesu”. Podobnie, jak w przypadku pierwszego apletu zacznę od deklaracji i definicji zmiennych.

```
25  int i=0,a=100,dyg=1,zwrot=5;
26  Image img,img1;
27  protected Graphics bufferGraphics;
28  protected Image offscreen;
29  int [] xPoints = {70,50,50,180,180,170};
30  int [] yPoints = {119,119,40,40,119,119};
31  int [] xStrzalka = {145,145,145,145};
32  int [] yStrzalka = {43,40,40,37};
33  int [] xObwod1 = {265,275,275,210,210,245};
34  int [] yObwod1 = {105,105,70,70,20,20};
35  int [] xObwod2 = {250,290,290,280,280,265};
36  int [] yObwod2 = {20,20,70,70,110,110};
37  char [] etykieta = {'A','I','U'};
```

Listing 12. Deklaracje i definicje zmiennych.

Na listingu pojawiły się nowe (względem poprzedniego apletu) zmienne. Związane jest to, z tym że aplety jednak nieco różnią się od siebie, bo w pierwszym mieliśmy rysowanie magnesu, a tutaj spotykamy się z elektromagnesem, gdzie mamy więcej

elementów do odrysowania. Znaczenie poszczególnych zmiennych wyjaśnię w trakcie omawiania poszczególnych metod. Zaglądając do załączonego kodu źródłowego można dopatrzyć się czterech metod, które wraz ze swą zawartością wyglądają identycznie, jak w pierwszym omawianym przeze mnie aplecie. I tak jest też z działaniem tych metod. Spełniają one identyczne zadania i dlatego nie będę już po raz kolejny ich omawiać tylko skieruję osoby zainteresowane do rozdziału „wsuwanie i wysuwanie magnesu". Tymi metodami są: „init()", „update()", „paint()", „paintComponent()" oraz „Uruchom()".

Ostatnia metoda, którą jest metoda „rysuj()" wymaga już krótkiego komentarza. Przy jej omawianiu ważne jest by czytelnik był jednak zaznajomiony z rozdziałem „wsuwanie i wysuwanie magnesu", gdyż omówiłem tam działanie metod, które służą do rysowania po danym kontekście graficznym. Jedyną wykorzystaną metodą, której wcześniej nie omówiłem jest metoda „drawArc()", która rysuje łuk. Przyjmuje ona sześć parametrów. Pierwsze dwa określają położenie łuku, kolejne szerokość i wysokość, dalej jest kąt początkowy, a ostatni parametr podaje kąt, o jaki oparty jest łuk względem kąta początkowego.

Zanim omówię metodę „rysuj()" chcę jednak zaznaczyć już na początku, że zwroty prądu w obwodzie ze zwojnicą, a tym samymi kierunek wychylenia wskazówki amperomierza są ustalone zgodnie z prawami fizyki, o czym wspomniałem w rozdziale trzecim poświęconym zagadnieniom fizycznym dotyczącym apletu.

Ciało metody „rysuj()" wygląda następująco:

```
86  public void rysuj(Graphics2D g){
87          //rysowanie magnesu
88          g.drawImage(img1, 60, 90, this);
89          g.setColor(Color.gray);
90          g.fill3DRect(165-i, 95, 100, 20, true);
91
92          g.setColor(Color.red);
93          for (int j=0;j<9;j++)
94          {
95              g.drawLine(173-i+j*10, 115,178-i+j*10, 95);
96              g.drawArc(178-i+j*10, 93, 3, 3, 0, 180);
97              g.drawArc(170-i+j*10, 113, 3, 3, 0, -180);
98          }
```

```
99
100     //obwód
101     g.drawImage(img, 60, 90, this);
102     g.setColor(Color.gray);
103     g.drawPolyline(xPoints, yPoints, 6);
104
105     //obwód elektromagnesu
106     g.setColor(new Color(240,240,240));
107     xObwod1[0]=xObwod2[5]=265-i;
108     g.drawImage(img, 60, 90, this);
109     g.setColor(Color.red);
110     g.drawPolyline(xObwod1,yObwod1, 6);
111     g.drawPolyline(xObwod2,yObwod2, 6);
112     g.drawLine(245, 10, 245, 30);
113     g.drawLine(250, 15, 250, 25);
114     g.drawChars(etykieta, 2, 1, 249, 13);
115
116     //tarcza
117     g.setColor(new Color(240,240,240));
118     g.fillOval(100,28, 25, 25);
119     g.setColor(new Color(120,120,120));
120     g.drawOval(100,28, 25, 25);
121     g.fill3DRect(112, 29, 2, 4, true);
122     g.drawLine(103, 32, 106, 35);
123     g.drawLine(122, 32, 119, 35);
124
125     g.setColor(Color.red);
126     g.setFont(new Font("Times New Roman",1,17));
127     g.drawChars(etykieta, 0,1,107,47);
128
129     g.setColor(Color.black);
130     g.setFont(new Font("Comic Sans MS",1,12));
131     g.drawChars(etykieta, 1,1,142,35);
132
```

```
133        g.setColor(new Color(120,120,120));
134        if (a==0)
135            g.drawLine(112,45,106+dyg,35-dyg); //wskazówka lewo
136        else
137            g.drawLine(112,45,120-dyg,35+dyg); //wskazówka prawo
138        xStrzalka[1]=145+zwrot;
139        xStrzalka[2]=145+zwrot;
140        g.drawPolyline(xStrzalka, yStrzalka, 4); //rysowanie strzałki
141    }
```

Listing 13. Metoda rysuj().

Polecenia 88-90:

Służą rysowaniu rdzenia elektromagnesu. Zmienna „i” odpowiada za przesuwanie.

Polecenia 92-98:

Służą do narysowania (stosując pętlę „for” od „0” do „8”) wszystkich zwojów na rdzeniu. Zmienna „i” j.w..

Polecenia 101-103:

Rysują szary obwód ze zwojnicą.

Polecenia 106-114

Odrysowują czerwony obwód zasilający elektromagnes. Zmienna „i” powoduje wydłużanie i skracanie czerwonych przewodów doprowadzonych do rdzenia.

Polecenia 117-123

Odpowiadają za tarczę amperomierza.

Polecenia 125-127 oraz 129-131:

Podpisują etykietami amperomierz i zwrot prądu.

Polecenia 133-137:

Rysują i sterują wskazówką amperomierza.

Polecenia 138-140:

Rysują odpowiednio strzałkę zwrotu prądu.

5.3. Włączanie i wyłączanie prądu w zwojnicy elektromagnesu.

Aplet ten różni się nieco od pozostałego tym, że elektromagnes jest nieruchomy i umieszczony w zwojnicy. Wszelkie zmieniające się obrazy są widoczne jedynie na amperomierzu oraz na przełączniku, który został dodany do czerwonego obwodu zasilającego elektromagnes. W momencie włączenia lub wyłączenia obserwuje się chwilowe wychylenie wskazówki raz w lewo, a raz w prawo w zależności od tego czy obwód był zamykany czy otwierany. Mimo tych zmian kilka metod pozostało niezmienionych i są to metody „init()”, „update()”, „paint()”, „paintComponent()” oraz „Uruchom()”. Zauważmy również, że metody te pozostały niezmienione od czasu opisywania pierwszego apletu i tam też odsyłam czytelnika, w celu zapoznania się z nimi i ze sposobem ich działania.

Zmiennymi, które zostały utworzone na potrzeby aplikacji są:

```
25  int i=0,a=100,dyg=1,zwrot=5,czas=0;
26  Image img,img1;
27  protected Graphics bufferGraphics;
28  protected Image offscreen;
29  int [] xPoints = {70,50,50,180,180,170};
30  int [] yPoints = {119,119,40,40,119,119};
31  int [] xStrzalka = {145,145,145,145};
32  int [] yStrzalka = {43,40,40,37};
33  int [] xObwod1 = {165,225,225,210,210,225};
34  int [] yObwod1 = {105,105,70,70,20,20};
35  int [] xObwod2 = {270,290,290,230,230,165};
36  int [] yObwod2 = {20,20,70,70,110,110};
37  char [] etykieta = {'A','I','U'};
```

Listing 14. Definicje i deklaracje zmiennych.

Część z powyższych zmiennych również jest taka, jak w poprzednich apletach. Natomiast o pozostałych zmiennych wspomnę przy opisywaniu metody „rysuj()”, gdyż jest ona jedyną metodą, która wymaga omówienia, w tym rozdziale. Chcę jednak

zaznaczyć już na początku, że zwroty prądu w obwodzie ze zwojnicą, a tym samymi kierunek wychylenia wskazówki amperomierza są ustalone zgodnie z prawami fizyki, o czym wspomniałem w rozdziale poświęconym zagadnieniom fizycznym dotyczącym apletu.

Metoda „rysuj()”:

```
85  public void rysuj(Graphics2D g){
86          //rysowanie magnesu
87          g.drawImage(img1, 60, 90, this);
88          g.setColor(Color.LIGHT_GRAY);
89          g.fill3DRect(65, 95, 100, 20, true);
90
91          g.setColor(Color.red);
92          for (int j=0;j<9;j++)
93          {
94                  g.drawLine(70+j*10, 115,78+j*10, 95);
95                  g.drawArc(78+j*10, 93, 3, 3, 0, 180);
96                  g.drawArc(70+j*10, 113, 3, 3, 0, -180);
97          }
98          //obwód
99          g.drawImage(img, 60, 90, this);
100         g.setColor(Color.gray);
101         g.drawPolyline(xPoints, yPoints, 6);
102
103         //obwód elektromagnesu
104         g.setColor(new Color(240,240,240));
105         g.drawImage(img, 60, 90, this);
106         g.setColor(Color.red);
107         g.drawPolyline(xObwod1,yObwod1, 6);
108         g.drawPolyline(xObwod2,yObwod2, 6);
109         g.drawLine(225, 10, 225, 30);
110         g.drawLine(230, 15, 230, 25);
111         g.drawChars(etykieta, 2, 1, 229, 13);
112         g.drawLine(230, 20, 250, 20);
```

```
113     g.drawOval(250, 17,3,3);
114     g.drawOval(270, 17,3,3);
115     g.drawLine(250,17,268+zwrot,12+zwrot); //przełącznik
116
117     //tarcza
118     g.setColor(new Color(240,240,240));
119     g.fillOval(100,28, 25, 25);
120     g.setColor(new Color(120,120,120));
121     g.drawOval(100,28, 25, 25);
122     g.fill3DRect(112, 29, 2, 4, true);
123     g.drawLine(103, 32, 106, 35);
124     g.drawLine(122, 32, 119, 35);
125
126     g.setColor(Color.red);
127     g.setFont(new Font("Times New Roman",1,17));
128     g.drawChars(etykieta, 0,1,107,47);
129
130     g.setFont(new Font("Comic Sans MS",1,12));
131     g.setColor(Color.black);
132     if ((a==50)&&(i<=8))
133     {
134             g.drawLine(112,45,120-dyg,35+dyg); //wskazówka prawo
135             g.drawChars(etykieta, 1,1,142,36);
136             g.setColor(new Color(120,120,120));
137             xStrzalka[1]=145+zwrot;
138             xStrzalka[2]=145+zwrot;
139             g.drawPolyline(xStrzalka, yStrzalka, 4); //rysowanie strzałki
140     }
141     else
142     if ((a==0)&&(i>=42))
143     {
144             g.drawLine(112,45,106+dyg,35-dyg);
145             g.drawChars(etykieta, 1,1,142,36);
146             g.setColor(new Color(120,120,120));
```

```
147             xStrzalka[1]=145+zwrot;
148             xStrzalka[2]=145+zwrot;
149             g.drawPolyline(xStrzalka, yStrzalka, 4); //rysowanie strzałki
150         }
151         else
152         {
153             g.drawLine(112,45,112,35);
154             g.drawLine(113,45,113,35);
155         } //wskazówka zero
156     }
```

Listing 15. Metoda rysuj().

Polecenia 87-89:

Rysowanie rdzenia elektromagnesu.

Polecenia 91-97:

Rysowanie wszystkich zwojów na elektromagnesie.

Polecenia 99-101:

Rysowanie szarego obwodu ze zwojnicą.

Polecenia 104-115:

Rysowanie czerwonego obwodu zasilającego elektromagnes, gdzie trzy ostatnie wiersze odpowiadają za odrysowanie przełącznika z okrągłymi stykami. Otwarcie i zamknięcie przełącznika jest ustalane przez wartość zmiennej zwrot, która modyfikuje odpowiednie współrzędne przełącznika.

Polecenia 118-124:

Rysują tarczę amperomierza.

Polecenia 126-128:

Podpisują etykietą amperomierz.

Polecenia 130-131:

Zmieniają kolor rysowania i ustawiają nową czcionkę.

Polecenia 134-139:

Rysują wskazówkę amperomierza wychyloną przez chwilę w prawo tuż po zamknięciu obwodu zasilającego elektromagnes oraz odpowiednio skierowaną strzałkę zwrotu prądu z etykietą. Sprawdzane są wartości zmiennych „a” i „i”, dzięki którym wiemy, kiedy obwód był otwarty, a kiedy zamknięty.

Polecenia 144-149:

Rysują wskazówkę amperomierza wychyloną przez chwilę w lewo tuż po otwarciu obwodu zasilającego elektromagnes oraz odpowiednio skierowaną strzałkę zwrotu prądu z etykietą. Sprawdzane są wartości zmiennych „a” i „i”, dzięki którym wiemy, kiedy obwód był otwarty, a kiedy zamknięty.

Polecenia 153 -154:

Rysowanie pionowej wskazówki, kiedy nic się nie dzieje.

5.4. Elektromagnes z opornicą suwakową.

To już ostatni aplet prezentujący sposób wzbudzania prądu indukcyjnego. Podobnie, jak w pozostałych apletach, tak i w tym istnieją metody, które są identyczne. Dla przypomnienia są nimi metody: „init()”, „update()”, „paint()”, „paintComponent()” oraz „Uruchom()”. W celu zapoznania się z ich działaniem odsyłam do rozdziału opisującego pierwszy aplet. Sposób przedstawiania obecnego apletu będzie analogiczny do pozostałych, czyli zacznę od prezentacji zmiennych (Listing 16.).

```
25  int i=0,a=100,dyg=1,zwrot=5,czas=0;
26  Image img,img1;
27  protected Graphics bufferGraphics;
28  protected Image offscreen;
29  int [] xPoints = {70,50,50,180,180,170};
30  int [] yPoints = {119,119,40,40,119,119};
31  int [] xStrzalka = {145,145,145,145};
32  int [] yStrzalka = {43,40,40,37};
```

```
33  int [] xObwod1 = {165,225,225,210,210,225};
34  int [] yObwod1 = {105,105,70,70,20,20};
35  int [] xObwod2 = {270,290,290,230,230,165};
36  int [] yObwod2 = {20,20,70,70,110,110};
37  int [] xPot = {240,240,250,250};
38  int [] yPot = {20,7,7,15};
39  int [] xStrzalka1 = {247,250,253};
40  int [] yStrzalka1 = {11,14,11};
41  char [] etykieta = {'A','I','U'};
```

Listing 16. Definicje i deklaracje zmiennych.

Jak widać na listingu (Listing 16.), oprócz stale przewijających się we wcześniejszych apletach zmiennych, pojawia się szereg nowych. I jak zwykłem omówię je za chwilę przy opisywaniu poleceń metody „rysuj()” z listingu (Listing 17.). Metodę tą przedstawię w taki sam sposób, jak wcześniej, czyli najpierw pojawi się listing zawierający ciało tej metody, a następnie grupami opiszę, co robią dane polecenia.

```
90  public void rysuj(Graphics2D g){
91          //rysowanie magnesu
92          g.drawImage(img1, 60, 90, this);
93          g.setColor(Color.LIGHT_GRAY);
94          g.fill3DRect(65, 95, 100, 20, true);
95
96          g.setColor(Color.red);
97          for (int j=0;j<9;j++)
98          {
99              g.drawLine(70+j*10, 115,78+j*10, 95);
100             g.drawArc(78+j*10, 93, 3, 3, 0, 180);
101             g.drawArc(70+j*10, 113, 3, 3, 0, -180);
102         }
103
104         //obwód
105         g.drawImage(img, 60, 90, this);
106         g.setColor(Color.gray);
```

```
107     g.drawPolyline(xPoints, yPoints, 6);
108     g.drawImage(img, 60, 90, this);
109
110     //obwód elektromagnesu
111     g.setColor(Color.red);
112     g.drawPolyline(xObwod1,yObwod1, 6);
113     g.drawPolyline(xObwod2,yObwod2, 6);
114     g.drawLine(225, 10, 225, 30);
115     g.drawLine(230, 15, 230, 25);
116     g.drawChars(etykieta, 2, 1, 229, 13);
117     g.drawLine(230, 20, 240, 20);
118
119     g.draw3DRect(250, 15, 20, 10, true); //potencjometr
120     xPot[2]=xPot[3]=250+i/2;
121     g.drawPolyline(xPot,yPot,4);
122
123     xStrzalka1[0]=247+i/2;
124     xStrzalka1[1]=250+i/2;
125     xStrzalka1[2]=253+i/2;
126     g.drawPolyline(xStrzalka1,yStrzalka1,3);
127
128     //tarcza
129     g.setColor(new Color(240,240,240));
130     g.fillOval(100,28, 25, 25);
131     g.setColor(new Color(120,120,120));
132     g.drawOval(100,28, 25, 25);
133     g.fill3DRect(112, 29, 2, 4, true);
134     g.drawLine(103, 32, 106, 35);
135     g.drawLine(122, 32, 119, 35);
136
137     g.setColor(Color.red);
138     g.setFont(new Font("Times New Roman",1,17));
139     g.drawChars(etykieta, 0,1,107,47);
140
```

```
141         g.setFont(new Font("Comic Sans MS",1,12));
142         g.setColor(Color.black);
143         if (a==0)
144             g.drawLine(112,45,106+dyg,35-dyg); //wskazówka lewo
145         else
146             g.drawLine(112,45,120-dyg,35+dyg); //wskazówka prawo
147         xStrzalka[1]=145+zwrot;
148         xStrzalka[2]=145+zwrot;
149         g.drawPolyline(xStrzalka, yStrzalka, 4); //rysowanie strzałki
150         g.drawChars(etykieta, 1,1,142,36);
151     }
```

Listing 17. Metoda rysuj().

Polecenia 92-94:

Rysują rdzeń elektromagnesu.

Polecenia 96-102:

Rysują wszystkie zwoje na elektromagnesie.

Polecenia 105-108:

Odrysowują obwód ze zwojnicą.

Polecenia 111-117:

Zajmują się narysowaniem czerwonego obwodu elektromagnesu korzystając z odpowiednich zmiennych.

Polecenia 119-121:

Odrysowują potencjometr ze wskaźnikiem, którego położenie reguluje zmienna „i”.

Polecenia 123-126:

Rysują strzałkę wskaźnika na potencjometrze sterowaną zmienną „i”.

Polecenia 129-135:

Rysują tarczę amperomierza.

Polecenia 137-139:

Etykietują amperomierz.

Polecenia 141-146:

Zmieniają kolor i czcionkę oraz rysują wskazówkę skierowaną w lewo, gdy zmniejszany jest opór na potencjometrze (zmienna „a”=0) i w prawo, gdy zwiększany jest opór na potencjometrze.

Polecenia 147-150:

Rysują strzałkę wskazującą zwrot prądu w obwodzie ze zwojnicą oraz zostaje nadana etykieta tej strzałce.

5.5. Ramka wchodząca w pole magnetyczne.

Jest to kolejny aplet, który symuluje wybrany problem fizyczny. Tutaj jest to ramka wchodząca w pole magnetyczne. Do tej pory, w uprzednio omawianych przeze mnie apletach można było zapoznać się z różnymi sposobami buforowania obrazu. W tym aplecie posłużyłem się inną metodą, która zapobiega migotaniu obrazu w trakcie działania symulacji. Do tego celu utworzyłem nową klasę, która rozszerza właściwości klasy „jPanel” (Listing 18.). Obiekt tej klasy jest takim ogólnym kontenerem (panelem), w którym można umieszczać inne obiekty np. przyciski, listy, pole tekstowe i inne kontrolki.

```
14  public class DrawingPanel extends JPanel {
15          java.awt.image.BufferedImage  buf;
16
17          /** Odrysowywanie panelu
18          *@param g obszar do rysowania
19          */
20          @Override
21          public void paintComponent (java.awt.Graphics g)
22          {
23                  super.paintComponent(g);
```

```
24                  g.drawImage (buf, 0, 0, this);
25          }
26  }
```

Listing 18. Tworzenie nowego panelu do rysowania.

Ciało rozszerzonej klasy „jPanel” widzimy powyżej jako nową klasę nazwaną przez mnie „DrawingPanel”, gdyż będzie dawała nam możliwość rysowania po panelu. W wierszu nr 15 tworzony jest obraz buforowany klasy „BufferedImage”, na którym będziemy odmalowywać elementy, po czym gotowy obraz z bufora odmalujemy na właściwym kontekście graficznym. O pojęciu kontekstu graficznego wspominałem przy okazji omawiania wcześniejszych apletów, także zapraszam do lektury poprzednich rozdziałów.

Jeżeli na utworzonym obiekcie klasy „DrawingPanel” wywołamy metodę „repaint()” to jak wiadomo spowoduje to uruchomienie szeregu innych metod odrysowujących dany obiekt. W naszym przypadku jedną z nich jest metoda „paintComponent()”. W pierwszym swoim wierszu (wiersz 23) odwołuje się do nie przeciążonej metody „paintComponent()” i powoduje jej wykonanie. W kolejnym natomiast odrysowuje obraz z bufora na kontekście graficznym związanym z panelem. I to by było na tyle jeżeli chodzi o buforowanie.

Teraz przejdźmy do klasy głównej apletu, która będzie rysować ramkę wchodzącą w pole magnetyczne oraz wykresy. Patrząc na działający aplet widać trzy główne obszary, po których odbywa się rysowanie. Są to właśnie panele utworzone za pomocą klasy „DrawingPanel”. Zostały one dodane do apletu ręcznie (wpisane bezpośrednio w kodzie, a nie poprzez kontrolkę środowiska Netbeans). Ponieważ obiekty klasy „jPanel” w środowisku Netbeans dodawane są poprzez przeciągnięcie odpowiedniej kontrolki na powierzchnię apletu (w widoku projektu), a kod generowany jest automatycznie, to nie będę omawiał sposobu dodawania obiektów klasy „DrawingPanel”, gdyż kod wygląda analogicznie. Dla wygody zamiast ręcznego wpisywania kodu można najpierw dodać nową kontrolkę (klasę „DrawingPanel”) do palety istniejących już kontrolek i stamtąd gotową przeciągnąć na powierzchnię apletu. O sposobie dodawania nowych kontrolek do palety napisałem przy okazji omawiania środowiska Netbeans.

Jak w przypadku poprzednich opisów zacznę od omówienia metod użytych w programie. I tym samym pierwszą z nich będzie metoda „init()” (Listing 19.), od której wszystko się zaczyna. Oczywiście ciało tej metody zostało objęte blokiem try-catch, aby

zwrócić odpowiednią informację na wypadek błędów przy inicjalizowaniu elementów. Oprócz tego metoda „init()” będzie wykonywana w oddzielnym wątku. O powyższych dwóch elementach mówione już było w rozdziale o języku Java (Rozdział 2.1) oraz przy okazji omawiania pierwszego apletu (Rozdział 5.1).

```
28  public void init() {
29          try {
30              java.awt.EventQueue.invokeAndWait(new Runnable() {
31              public void run() {
32                      setSize(540, 400);
33                      initComponents();
34                      SpinnerNumberModel modelv=new
                        SpinnerNumberModel(0.1,0,20,0.01);
35                      SpinnerNumberModel modelB=new
                        SpinnerNumberModel(1.00,0.001,10,0.001);
36                      SpinnerNumberModel modelx=new SpinnerNumberModel(70,
                        1, 200, 1);
37                      SpinnerNumberModel modely=new SpinnerNumberModel(100,
                        1, 200, 1);
38                      jSpinner1.setModel(modelx);
39                      jSpinner2.setModel(modely);
40                      jSpinner4.setModel(modelv);
41                      jSpinner3.setModel(modelB);
42                      init_boki();
43                      rys_ramka();
44              }
45              });
46          } catch (Exception ex) {
47              ex.printStackTrace();
48          }
49  }
```

Listing 19. Metoda init().

W pierwszej kolejności ustawiane są wymiary apletu (wiersz 32). W dalszej kolejności (wiersz 33) następuje inicjalizowanie elementów tworzących aplet. W kodzie metody „initComponents()”, która najczęściej tworzona automatycznie w środowisku Netbeans odbywa się programowe (rzeczywiste) dodawanie tych elementów do apletu, które dodaliśmy w widoku projektu na jego powierzchnię, czyli w tym przypadku są to: trzy panele „DrawingPanel”, etykiety („jLabel”), przyciski („jButton”) oraz pola wyboru wartości dla odpowiednich wartości fizycznych („jSpinner”). W nawiasach podałem nazwy programowe, gdyż przy omawianiu kodu łatwiej będzie odwoływać się, do konkretnych elementów na aplecie. W wierszach 34-37 tworzone są modele pól wyboru, czyli określane są parametry możliwych do wyboru wartości za pomocą pól wyboru. Konstruktor klasy „SpinnerNumberModel” przyjmuje różne parametry. Ja jednak skupie się na tych wykorzystanych w programie. Na pierwszej pozycji podaje się wartość domyślną ustawioną na początku, następnie wartość minimalną, maksymalną oraz krok o jaki ma być zwiększana bądź zmniejszana ta wartość. Na przykład prędkość początkowa ruchu ramki została ustawiona na 0.1 m/s, a zakres, w jakim można regulować prędkość na wartości od 0 do 20 m/s, przy wartość można regulować z dokładnością do 0.01 m/s (SpinnerNumberModel modelv = new SpinnerNumberModel(0.1,0,20,0.01)).

W kolejnym wierszach (wiersze 38-41) za pomocą metody „setModel” ustawiane są pola wyboru wg odpowiednich modeli.

W dalszej kolejności wywoływana jest metoda „init_boki()” (Listing 20.). Metoda „init_boki()” inicjalizuje odpowiednie zmienne (długości boków ramki) odpowiednimi wartościami ustawionymi przy użyciu pól wyboru.

```
346  public void init_boki()
347  {
348      bokx=Double.valueOf(jSpinner1.getValue().toString());
349      boky=Double.valueOf(jSpinner2.getValue().toString());
350      intbokx = Integer.valueOf(jSpinner1.getValue().toString());
351      intboky = Integer.valueOf(jSpinner2.getValue().toString());
352  }
```

Listing 20. Metoda init_boki().

Zmienne te są liczbami typu „Double”, dlatego konieczna jest konwersja na ten typ. Metoda „Double.valueOf()” zwraca liczbę, która zapisana jest w postaci tekstu („string”)

jako parametr. Metoda „getValue()” pobiera wartość z pola wyboru „jSpinner” by następnie zwrócić wartość jej reprezentację tekstowa za pomocą metody „toString()”. Pojawia się pytanie czemu zamiast tych wszystkich przekształceń nie można napisać np. „bokx = jSpinner1.getValue()”. Dzieje się tak dlatego, że metoda „getValue()” zwraca obiekt typu „Object”, czyli wartość typu ogólnego dla wszystkich obiektów w języku Java. Dwa ostatnie wiersze są analogiczne tyle, że dotyczą liczb typu całkowitego („Integer”). Dwa różne typy zmiennych, które reprezentują długości boków zostały utworzone na potrzeby pozostałych metod, które będę wymagały odpowiedniego typu zmiennych.

Kolejną i ostatnią metodą uruchamianą w ciele metody „init()” jest metoda „rys_ramka()” (Listing 22.). Pierwszą wywoływaną w jej ciele metodą jest metoda „init_dp1()” (Listing 21.), w której inicjalizowana jest tylko zmienna będąca buforowanym obrazem należąca do panelu pierwszego, gdzie kolejne parametry to odpowiednio, szerokość, wysokość i typ obrazu. Określony tutaj typ („BufferedImage.TYPE_4BYTE_ABGR”) mówi nam, że jest to obraz, w którym wiedza o kolorach zapisywana jest w 4 bajtach, gdzie każdy bajt określa nasycenie koloru czerwonego (kanał R), zielonego (kanał G) i niebieskiego (kanał B), a czwarty bajt odpowiada za przeźroczystość (kanał alfa).

```
354  private void init_dp1()
355  {
356      dp1.buf = new BufferedImage(250, 250,
         BufferedImage.TYPE_4BYTE_ABGR);
357  }
```

Listing 21. Metoda init_dp1().

Wróćmy do naszej metody „rys_ramka()”. Poniżej umieszczam jej ciało, aby było łatwiej zobrazować, co się w niej dzieje.

```
444  private void rys_ramka()
445  {
446      init_dp1();
447      Graphics2D g1 = dp1.buf.createGraphics();
448      g1.setColor(Color.RED);
449      if (intbokx>intboky)
```

```
450         {
451                 g1.drawRect((int)((v*t)*100+(143-x*100))-65, 100, (int)(x*100),
                    (int)(y*50));
452                 g1.drawRect((int)((v*t)*100+(143-x*100))+1-65, 101,
                    (int)(x*100)-2, (int)(y*50)-2);
453         }
454         else
455                 if (intbokx==intboky)
456                 {
457                         g1.drawRect((int)((v*t)*100+(143-x*100))-65, 75,
                            (int)(x*100), (int)(x*110));
458                         g1.drawRect((int)((v*t)*100+(143-x*100))+1-65, 76,
                            (int)(x*100)-2, (int)(x*110)-2);
459                 }
460                 else
461                 {
462                         g1.drawRect((int)((v*t)*100+(143-x*100))-65, 60,
                            (int)(x*100), (int)(y*100));
463                         g1.drawRect((int)((v*t)*100+(143-x*100))+1-65, 61,
                            (int)(x*100)-2, (int)(y*100)-2);
464                 }
465         dp1.repaint();
466     }
```

Listing 22. Metoda rys_ramka().

Po wywołaniu metody „init_dp1()” na zainicjalizowanym w niej obrazku tworzony jest kontekst graficzny g1, na którym będzie rysowana poruszająca się ramka (wiersz 447). Kolor rysowania ustawiany jest na czerwony (wiersz 448). Dalej w zależności od tego, który bok jest dłuższy rysowana jest odpowiednich wymiarów ramka, tak by lepiej zobrazować relację pomiędzy bokami. W programie istnieją dwie metody, które czuwają nad wprowadzanymi przez użytkownika informacjami o długości i szerokości ramki. Są nimi odpowiednio metody „x_change()” oraz „y_change()” (Listing 23.).

```
583     private void x_change(javax.swing.event.ChangeEvent evt) {
```

```
584         init_boki();
585         rys_ramka();
586     }
587
588         private void y_change(javax.swing.event.ChangeEvent evt) {
589         init_boki();
590         rys_ramka();
591     }
```

Listing 23. Metod x_chane() i y_change().

Metody te są wywoływane za każdym razem, kiedy użytkownik zmienia daną wartość szerokości czy wysokości ramki i powodują odrysowanie ramki z nowymi parametrami zainicjalizowanymi w metodzie „init_boki()”.

Do określenia warunków instrukcji „if” przydały się wcześniej zainicjalizowane zmienne typu całkowitego „intbokx” i „intboky”. Za ruch ramki odpowiadają konkretne zmienne, które można dostrzec w parametrach rysowanego prostokąta. Zmienne te zmieniają swoją wartość w trakcie działanie animacji, tym samym zmieniają położenie rysowanego prostokąta. Na samym końcu (wiersz 466) wywoływana jest metoda „repaint()” na panelu „dp1”, która powoduje odmalowanie obrazu na właściwym kontekście graficznym, czyli na panelu. Sposób odmalowywania określony jest klasie „DrawingPanel”, którą omówiłem już wcześniej.

Metodą, która rozpoczyna ruch ramki oraz rysowanie wykresów jest metoda „jButton1ActionPerformed()”, która uruchamiana jest w chwili, gdy użytkownik naciśnie przycisk „Start”. Jednak zanim opiszę, co dzieje się w tej metodzie, przedstawię i omówię wszystkie metody, które są wywoływane w ciele tej metody. Jedną z nich jest metoda „ustawPoStarcie()” (Listing 24.), której już sama nazwa wskazuje na jej funkcjonalność.

```
542     private void ustawPoStarcie()
543     {
544         b=c=1;
545         start = true;
546         pauza = false;
547
548         bokx=Double.valueOf(jSpinner1.getValue().toString());
```

```
549         boky=Double.valueOf(jSpinner2.getValue().toString());
550         B=Double.valueOf(jSpinner3.getValue().toString());
551         V=Double.valueOf(jSpinner4.getValue().toString())*100;
552
553         jSpinner1.setEnabled(false);
554         jSpinner2.setEnabled(false);
555         jSpinner3.setEnabled(false);
556         jSpinner4.setEnabled(false);
557         jButton1.setText("Pause");
558     }
```

Listing 24. Metoda ustawPoStarcie().

W pierwszych jej wierszach aż do wiersza 552 ustawiane są odpowiednie zmienne sterujące, w kolejnych czterech blokowane są pola wyboru, by nie można ich modyfikować w trakcie ruchu ramki. Ostatni wiersz zmienia tekst na przycisku na „Pause”.

Kolejnymi metodami są „init_dp2()” (Listing 25.) oraz „init_dp3()”, które przygotowują układy współrzędnych do rysowania wykresów.

```
359     private void init_dp2()
360     {
361         dp2h = dp2.getHeight();
362         dp2w = dp2.getWidth();
363         dp2.buf = new BufferedImage(dp2w, dp2h,
            BufferedImage.TYPE_4BYTE_ABGR);
364         Graphics2D g = dp2.buf.createGraphics();
365         g.setColor(Color.BLACK);
366         g.drawLine(0, dp2h-1, dp2w, dp2h-1);
367         g.drawLine(0, 0, 0, dp2h-1);
368         //skala
369         g.drawLine(35,dp2h,35,dp2h-4);
370         g.drawLine(70,dp2h,70,dp2h-4);
371         g.drawLine(105,dp2h,105,dp2h-4);
372         g.drawLine(140,dp2h,140,dp2h-4);
373         g.drawLine(175,dp2h,175,dp2h-4);
```

```
374         g.drawLine(210,dp2h,210,dp2h-4);
375         g.drawLine(245,dp2h,245,dp2h-4);
376         g.drawLine(280,dp2h,280,dp2h-4);
377         g.drawLine(1,(int)(0.25*dp2h),4,(int)(0.25*dp2h));
378         g.drawLine(0,(int)((0.75/2+0.25)*dp2h),4,(int)((0.75/2+0.25)*dp2h));
379
380         //opis skali
381         czas=(float)(bokx/V);
382         NumberFormat nf = NumberFormat.getInstance(Locale.ENGLISH);
383         nf.setMaximumFractionDigits(1);
384         nf.setMinimumFractionDigits(1);
385         g.drawString(nf.format(czas), 65,dp2h-7);
386         czas=czas/2;
387         g.drawString(nf.format(czas), 30,dp2h-7);
388         g.drawString(nf.format(3*czas), 100,dp2h-7);
389         g.drawString(nf.format(4*czas), 135,dp2h-7);
390         g.drawString(nf.format(5*czas), 170,dp2h-7);
391         g.drawString(nf.format(6*czas), 205,dp2h-7);
392         g.drawString(nf.format(7*czas), 240,dp2h-7);
393
394         g.drawString(""+wartB,5,(int)(0.24*dp2h));
395         g.drawString(""+wartB/2,5,(int)(0.61*dp2h));
396         g.setColor(Color.BLUE);
397         dp2.repaint();
398     }
```

Listing 25. Metoda init_dp2().

Dwa pierwsze wiersze tej metody pobierają wysokość i szerokość drugiego panelu , na którym rysowany jest wykres strumienia magnetycznego. W dalszej kolejności inicjalizowany jest obraz buforowy o rozmiarach panelu. (wiersz 364) oraz tworzony jest na tym buforze nowy kontekst graficzny. Kolor rysowania ustawiany jest na „czarny” (wiersz 365). Wiersze 366-378 odpowiadają za rysowanie za pomocą linii układu współrzędnych wraz ze skalą.

W dalszej części kodu tworzony jest opis skali. Aby poprawnie były wyświetlane wartości czasu konieczne było utworzenie odpowiedniego formatu liczbowego za co odpowiada wiersz 382 oraz dwa kolejne, które ustawiają minimalne i maksymalna liczbę cyfr po przecinku na 1. W wierszach 387-396 rysowane są za pomocą metody „drawString()” odpowiednie wartości na skali wyznaczone na podstawie ustalonych rozmiarów ramki oraz pozostałych wartości fizycznych (np. czas liczony w wierszu 381 jest to iloraz długości boku x do prędkości, czyli zgodnie z zasadami fizyki). Odpowiednia sformatowaną wartość liczby zwraca metoda „format()” wywołana, z parametrem w postaci tej liczby, na danym formacie.

W ostatniej linijce następuję odmalowanie panelu.

Metody „init_dp3()” nie będę omawiał oddzielnie, gdyż jest ona metodą analogiczną do omówionej przed chwilą, z tym że odmalowywanie (w oparciu wartości odpowiednich zmiennych) odbywa się na trzecim panelu, gdzie rysowany będzie wykres SEM.

Inną ważną metodą jest metoda „skalaY()”. Odpowiada ona za ustawienie poprawnego wyświetlania wartości na osiach pionowych obu wykresów. To znaczy, że w zależności od tego, jakie wartości przyjmuje wartość strumienia indukcji magnetycznej, tak odpowiednio podpisywana jest oś np. jeżeli wartość strumienia jest większa lub równa 0.000001 i mniejsza od 0.001 to oś podpisywana tekstem: "Strumień [10^-3 Wb]". Analogicznie dzieje się dla wartość SEM. Dodatkowo stosuje się zaokrąglanie, aby wartości liczbowe wyświetlane przy osi były czytelne. W tym przypadku wartości zaokrąglane są do trzech miejsc po przecinku. Przy osi widać również wartości, które mają cztery miejsca po przecinku, ale jest to wynikiem dzielenia przez 2 wartości zaokrąglonej do trzech miejsc po przecinku. Za zaokrąglanie odpowiedzialna jest oczywiście metoda „zaokraglanie()” (Listing 26.), która jako parametry przyjmuje liczbę (typu „double”) do zaokrąglenia oraz wartość (typu „Integer”) mówiącą ile miejsc po przecinku ma mieć zaokrąglana liczba.

```
701  public double zaokraglanie(double liczba,int decimalPlace)
702  {
703      BigDecimal bd = new BigDecimal(liczba);
704      bd = bd.setScale(decimalPlace, BigDecimal.ROUND_HALF_DOWN);
705      return bd.doubleValue();
706  }
```

Listing 26. Metoda zaokraglanie().

Zaokrąglanie odbywa się w taki sposób, że z danej liczby typu „double” tworzona jest liczba typu „BigDecimal” (wiersz 703). Za pomocą metody „setScale()” wykonanej na nowej liczbie dokonywane jest zaokrąglenie (wiersz 704). Jako pierwszy parametr podaje się liczbę miejsc po przecinku, a drugim parametrem jest typ zaokrąglenia. Tutaj zaokrąglana jest liczba od połowy w dół np. liczba 6,2555 po zaokrągleniu do trzech miejsc po przecinku będzie równa 6,255. Metoda „zaokraglanie()” zwraca zaokrągloną wartość typu double (wiersz 705).

Innymi istotnymi metodami są metody, które odpowiadają za zatrzymywanie czy wznawianie ruchu ramki. I są nimi metody „ustawPoWznowieniu()” oraz „ustawPoStop()” (Listing 27.). Obie metody uruchamiane są po naciśnięciu przycisków znajdujących się na aplecie. Ich ciała wyglądają następująco.

```
560  private void ustawPoWznowieniu()
561  {
562      b=c=1;
563      jButton1.setText("Pause");
564      pauza = false;
565  }
566
567  public void ustawPoStop()
568  {
569      jButton1.setText("Start");
570      jLabel10.setText("0 Wb");
571      jLabel11.setText("0 V");
572      start = false;
573      pauza = false;
574      jSpinner1.setEnabled(true);
575      jSpinner2.setEnabled(true);
576      jSpinner3.setEnabled(true);
577      jSpinner4.setEnabled(true);
578      t = t1 = strum1 = 0;
579  }
```

Listing 27. Metody ustawPoWznowieniu() i ustawPoStop().

Patrząc na powyższy kod widzimy, że w obu metodach ustawiane są odpowiednie etykiety, teksty na przyciskach oraz zmienne. W metodzie „ustawPoStop()” odblokowywane są pola wyboru by można było ponownie ustawić wielkości ramki oraz jej prędkość oraz wartość indukcji elektromagnetycznej.

Ostatnimi dwiema metodami, które omówię są metody wywoływane bezpośrednio po naciśnięciu przycisku „Start/Pause” i „Stop” (Listing 28.).

```
// przycisk START/PAUSE
511     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
512         intbokx = Integer.valueOf(jSpinner1.getValue().toString());
513         intboky = Integer.valueOf(jSpinner2.getValue().toString());
514         rys_ramka();
515         if (start==false&&pauza==false)
516         {
517             ustawPoStarcie();
518             skalaY(); // ustawienie poprawnych wartości na osi Y
519             init_dp2();
520             init_dp3();
521             new Symulacja(this);
522         } else if(pauza==false)
523             {
524                 jButton1.setText("Start");
525                 pauza = true;
526             }else
527             {
528                 ustawPoWznowieniu();
529             }
530     }
531
532     //przycisk STOP
533     private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
534         ustawPoStop();
535         rys_ramka();
536         init_dp2();
537         init_dp3();
538     }
```

Listing 28. Metody dotyczące przycisku „Start/Pause” i „Stop”.

Pierwsza metoda inicjalizuje odpowiednie zmienne wartościami z pól wyboru (wiersze 513-514) po czym w kolejnym z wierszy wywołuje metodę „rys_ramka()”. W dalszej kolejności są instrukcje „if-else”. Jeżeli animacja była zatrzymana (po

naciśnięciu przycisku „STOP” lub po pierwszym uruchomieniu apletu), to wykonywany jest blok instrukcji 518-522. Jeżeli ramka porusza się, to wykonywany jest blok 525-526, a jeżeli ruch ramki był wstrzymany, to wywoływana jest metoda „ustawPoWznowieniu()” (wiersz 529).

Metoda wywołana przez naciśnięcie przycisku „Stop” powoduje zatrzymanie animacji poprzez wykonaniu bloku instrukcji 535-538. Wszystkie metody, które pojawiają się w ciele tych dwóch metod związanych z przyciskami omówiłem powyżej. Jedyne zaskoczenie może wywołać instrukcja w wierszu 522 („new Symulacja(this)”).

Instrukcja ta powoduje utworzenie nowej symulacji, a dokładnie obiektu klasy „Symulacja”, który tak naprawdę utworzy nowy wątek w ramach którego będą wykonywały się instrukcje poruszające ramką oraz odpowiadające za rysowanie wykresów. Ze względu na to, że kod definiujący klasę „Symulacja” jest bardzo długi, nie zamieszczę go w opisie, ponieważ zająłby kilka stron. Zainteresowanych odsyłam do załącznika. Klasa ta jest w nim szczegółowo opisana za pomocą komentarzy, które powinny w jasny i zrozumiały dla czytelnika sposób wyjaśnić, co dzieje się w jej wnętrzu. Osoby, które zapoznały się z wcześniejszymi rozdziałami, a w szczególności z rozdziałem o języku Java nie powinny mieć żadnej trudności w zrozumieniu kodu powyższej klasy. Tutaj powiem tylko tyle, że tworzony w niej wątek wykonuje na bieżąco (w pętli) obliczenia strumienia magnetycznego i SEM w danej chwili i na podstawie tych obliczeń odrysowuje na odpowiednich panelach wykresy strumienia magnetycznego i SEM. Pętla „do-while” sterowana jest stanem zmiennych „pauza” i „start”, które powodują wstrzymanie lub całkowite zatrzymanie wykonywania się pętli. Wartości tych zmiennych są uzależnione od przycisków naciskanych przez użytkownika.

5.6. Ramka obracająca się w polu magnetycznym.

W aplecie tym wykrzystywana jest identyczna klasa buforująca rozszerzająca klasę „Applet” („public class BufferedApplet extends Applet”). Nie będę jej w takim razie tutaj omawiał, a zainteresowanych odsyłam do rozdziału „ruch obrazka”.

Jak zwykle na początku zajmę się omówieniem metody „init()”. Oto jej ciało:

```
41  public void init() {
42          try {
43              java.awt.EventQueue.invokeAndWait(new Runnable() {
```

```
44            public void run() {
45                setSize(540, 300);
46                setLayout(new FlowLayout(FlowLayout.LEFT,40,250));
47                przycisk1 =  new JButton("=START=");
48                przycisk1.addActionListener(new  java.awt.event.ActionListener()
    {
49                public void actionPerformed(java.awt.event.ActionEvent evt) {
50                    przycisk1ActionPerformed(evt);
51                }
52                });
53                add(przycisk1);
54                slider = new JSlider();
55                slider.setMinimum(1);
56                slider.setMaximum(6);
57                slider.setBackground(new Color(0,150,0,255));
58                add(slider);
59                setBackground(new Color(0,150,0,255));
60                }
61            });
62        } catch (Exception ex) {
63            ex.printStackTrace();
64        }
65  }
```

Listing 29. Metoda init().

Oczywiście, jak we wcześniejszych apletach blok instrukcji zawartych w ciele tej metody jest blokiem „try-catch”, wykonywanym w oddzielnym wątku. Pierwszą instrukcją kształtująca powierzchnię apletu jest oczywiście instrukcja ustalająca wymiary apletu (wiersz 45). Nie znaną do tej pory instrukcję zawiera wiersz 46. Ustawiany jest tutaj wzorzec apletu „FlowLayout”. Wzorzec odpowiada za rozmieszczenie komponentów na kontenerze, czyli w naszym przypadku na aplecie. Konstruktor klasy „FlowLayout” przyjmuje tu trzy parametry. Pierwszy odpowiada za wyrównanie elementów, które może odbywać się na pięć różnych sposobów: do lewej krawędzi, do prawej krawędzi, wyśrodkowanie oraz do lewej i prawej krawędzi w zależności od orientacji apletu. Kolejne

dwa określają odpowiednio poziome i pionowe odstępy pomiędzy komponentami a krawędziami oraz samymi komponentami. Wiersz 47 wypisuje tekst „=START=” na przycisku. W kolejnych pięciu wierszach (wiersze 48-52 dodają do przycisku tzw. interfejs słuchacza, czyli taki obiekt, który nasłuchuje czy na przycisku nie są wykonywane żadne akcje, np. czy przycisk nie został naciśnięty albo czy nie znajduje się nad nim kursor. Jest wiele różnorodnych akcji, które mogą być odbierane przez takiego „słuchacza”. Kiedy zostanie wykonana na przycisku akcja „słuchacz” spowoduje wykonanie metody „przycisk1ActionPerformed()”, która zdefiniowana jest w dalszej części programu, którą omówię za chwilę. Kolejnym krokiem jest dodanie tak przygotowanego przycisku do powierzchni apletu (wiersz 53). W wierszu 54 tworzony jest potencjometr, który służy do regulacji prędkości obracania się ramki. Kolejne wiersze to ustalenie wartości minimalnej na potencjometrze na wartość „1”, a wartości maksymalnej na wartość „6”. Krok zmiany wartość jest domyślnie ustawiony na „1”. Zanim potencjometr zostaje dodany do powierzchni apletu (wiersz 58) ustawiany jest jeszcze kolor jego tła (wiersz 57). Ostatni wiersz ustawia kolor tła całego apletu i na tym kończy się omawianie metody „init()”.

Teraz zajmę się już wspomnianą przeze mnie metodą „przycisk1ActionPerformed()” (Listing 30.).

```
67  private void przycisk1ActionPerformed(ActionEvent evt) {
68          if (start)
69          {
70              start=false;
71              Uruchom();
72              przycisk1.setText("PAUSE");
73          }else
74          {
75              start=true;
76              przycisk1.setText("=START=");
77          }
78  }
```

Listing 30. Metoda obsługująca przycisk.

Kiedy użytkownik naciśnie przycisk „=START=” wartość zmiennej „start” ustawiana jest na „false”, wywoływana jest metoda „Uruchom()” oraz na przycisku

pojawia się napis „PAUSE”, co oznacza z kolei, że użytkownik ma możliwość wstrzymania obrotu ramki w dowolnej chwili. Kiedy tak się stanie, zmienna „start” ustawiana jest na „true” i na przycisku pojawia się napis „=START=” umożliwiający wznowienie ruchu obrotowego.

Zadaniem metody „Uruchom()” (Listing 31.) jest utworzenie nowego wątku, w którym będzie starował cyklicznym wykonywaniem przeciążonej metody „paint()”, która odrysowuje aplet i jego elementy. O wątkach czytaj w rozdziale o języku Java (Rozdział 2.1) oraz w pierwszym aplecie (Rozdział 5.1).

```
250 public void Uruchom()
251 {
252     Timer latanie = new Timer();
253     latanie.schedule(new TimerTask(){
254
255         @Override
256         public void run() {
257             i++;
258             f=slider.getValue();
259             a=Math.sin(Math.PI/120*i*f);
260             yaxis=(int)(r*a);
261             y1axis=(int)(rr*a);
262             if (a==1)
263                 nr=0;
264             else
265                 if (a==-1)
266                     nr=1;
267             repaint();
268             if (start==true) this.cancel();
269         }
270     }, 0,50);
271 }
```

Listing 31. Metoda Uruchom().

W wierszu 258 do zmiennej „f” pobierana jest wartość prędkości z potencjometru, aby na jej podstawie wyliczyć sinus kąta, o jaki obraca się ramka w kolejnych krokach „i”. W wierszach 260-261 obliczane są wartości, które modyfikują odpowiednie współrzędne rysowanej ramki tak, by na powierzchni apletu był widoczny efekt obrotu ramki. Instrukcje warunkowe „if” w wierszach 262-266 pilnują półobrotów ramki, gdyż w każdym półobrocie w inny sposób modyfikowane są współrzędne ramki (krawędź pozioma raz skraca się a raz wydłuża). Po tych instrukcjach wywoływana jest metoda „paint()” poprzez wywołanie metody „repaint()” (wiersz 267). Po odmalowaniu apletu sprawdzane jest czy użytkownik nie wstrzymał obrotu. Jeśli tak to następuje zakończenie wątku, a jeśli nie to rozpoczyna się kolejna pętla.

Pozostała jeszcze tylko jedna metoda do omówienia i jest nią oczywiście przeciążona metoda „paint()” (Listing 32.).

```
81  public void paint(Graphics g)
82  {
83          int x, x0,xV0;
84          double y0, y,yV,yV0, A,  t, offset;
85          A = (double) height / 4;
86          offset = (double) height / 2;
87          x0 = 0;
88          xV0=0;
89          y0 = offset;
90          yV0=offset;
91          super.paint(g);
92          g.setFont(new Font("Comic Sans MS",Font.PLAIN,12));
93          g.drawString("Regulator prędkości obrotu ramki", 170, 252);
94          int [] xPoints={0,0,0,0,0,0,0,0};
95          if(nr==1)
96          {
97              xPoints[0]=mx-yaxis/4;
98              xPoints[1]=mx+140+yaxis/4;
99              xPoints[2]=mx+140-y1axis/4;
100             xPoints[3]=mx+170+y1axis/4;
101             xPoints[4]=mx+170+y1axis/4;
```

```
102             xPoints[5]=mx+140-y1axis/4;
103             xPoints[6]=mx+140-yaxis/4;
104             xPoints[7]=mx+yaxis/4;
105         }else
106         {
107             xPoints[7]=mx-yaxis/4;
108             xPoints[6]=mx+140+yaxis/4;
109             xPoints[5]=mx+140-y1axis/4;
110             xPoints[4]=mx+170+y1axis/4;
111             xPoints[3]=mx+170+y1axis/4;
112             xPoints[2]=mx+140-y1axis/4;
113             xPoints[1]=mx+140-yaxis/4;
114             xPoints[0]=mx+yaxis/4;
115         }
116         int [] yPoints =
            {my+Math.abs(yaxis),my+Math.abs(yaxis),my+65+Math.abs(y1axis),
117         my+65+Math.abs(y1axis),my+75-Math.abs(y1axis),my+75-
            Math.abs(y1axis),
118         my+140-Math.abs(yaxis),my+140-Math.abs(yaxis)};
119
120         g.setColor(Color.red);
121         g.fillRect(mx-40, 220, 220, 15); //pasek czerwony
122         int [] a2={mx-10,mx+150,mx+180,mx-40};
123         int [] b2={my+125,my+125,my+160,my+160};
124         g.fillPolygon(a2,b2,4);
125
126         g.setColor(Color.blue);
127         g.fillRect(mx-40, 20, 220, 15); //pasek niebieski
128         int [] a1={mx-10,mx+150,mx+180,mx-40};
129         int [] b1={my+10,my+10,my-25,my-25};
130         g.fillPolygon(a1,b1,4);
131
132         g.setColor(Color.black);
133         g.drawRect(mx-40, 20, 220, 15);
```

```
134     g.drawRect(mx-40, 220, 220, 15);
135     g.drawPolygon(a1,b1,4);
136     g.drawPolygon(a2,b2,4);
137     g.setColor(Color.white);
138     g.drawString("N", 140, 32);
139     g.drawString("S", 140, 232);
140
141     //przewody
142     g.setColor(new Color(254,248,180));
143     g.fillRect(255, 125, 100, 3);
144     g.fillRect(255, 133, 100, 3);
145
146     //podłączenia
147     g.setColor(new Color(40,40,40));
148     g.fillRoundRect(320, 121, 10, 9,1,1);
149     g.fillRoundRect(320, 132, 10, 9,1,1);
150
151     //linie pola
152     g.setColor(Color.white);
153     g.fill3DRect(75, 56, 1, 140,true);
154     g.fill3DRect(105, 56, 1, 140,true);
155     g.fill3DRect(135, 56, 1, 140,true);
156     g.fill3DRect(165, 56, 1, 140,true);
157     g.fill3DRect(195, 56, 1, 140,true);
158     g.fill3DRect(225, 56, 1, 140,true);
159     g.drawLine(75,140,73,135);
160     g.drawLine(75,140,77,135);
161     g.drawLine(105,140,103,135);
162     g.drawLine(105,140,107,135);
163     g.drawLine(135,140,133,135);
164     g.drawLine(135,140,137,135);
165     g.drawLine(165,140,163,135);
166     g.drawLine(165,140,167,135);
167     g.drawLine(195,140,193,135);
```

```
168     g.drawLine(195,140,197,135);
169     g.drawLine(225,140,223,135);
170     g.drawLine(225,140,227,135);
171
172     g.setColor(Color.orange);
173     g.drawPolygon(xPoints, yPoints, 8); //ramka
174     g.setColor(Color.DARK_GRAY);
175     g.fillRoundRect(170+mx, 60+my, 10, 20, 2, 2); //klocek
176
177     //linie pola
178     g.setColor(Color.white);
179     g.fill3DRect(50, 41, 1, 175,true);
180     g.fill3DRect(90, 41, 1, 175,true);
181     g.fill3DRect(130, 41, 1, 175,true);
182     g.fill3DRect(170, 41, 1, 175,true);
183     g.fill3DRect(210, 41, 1, 175,true);
184     g.fill3DRect(250, 41, 1, 175,true);
185     g.drawLine(50,160,48,155);
186     g.drawLine(50,160,52,155);
187     g.drawLine(90,160,88,155);
188     g.drawLine(90,160,92,155);
189     g.drawLine(130,160,128,155);
190     g.drawLine(130,160,132,155);
191     g.drawLine(170,160,168,155);
192     g.drawLine(170,160,172,155);
193     g.drawLine(210,160,208,155);
194     g.drawLine(210,160,212,155);
195     g.drawLine(250,160,248,155);
196     g.drawLine(250,160,252,155);
197     //miernik
198     g.setColor(new Color(48,48,48));
199     g.fillRoundRect(290+x0, (int) y0-65, 240,190,3,3);
200     g.setColor(new Color(40,40,40));
201     g.fillRoundRect(310+x0, (int) y0+125, 40,20,2,2);
```

```
202         g.fillRoundRect(470+x0, (int) y0+125, 40,20,2,2);
203
204         g.setColor(Color.gray);
205         g.fillRoundRect(305, 21,211,169,3,3); //ekran
206         g.setColor(Color.green);
207         g.fillRect(309, 24,203,163);
208         //wykres
209
210         g.setColor(Color.black);
211         g.drawLine (311+width/2, 25, 311+width/2,30+ height); //linia pionu
212         g.drawLine (310+x0, 31+(int) y0, 310+width,31+ (int) y0); //linia
    poziomu
213         for (int j=0;j<11;j++)
214             g.drawLine (210+width/2+j*20, 25, 210+width/2+j*20,35+ height);
                //linia pionu
215         for (int j=0;j<9;j++)
216             g.drawLine (310+x0, (int) y0-50+j*20, 310+width,(int) y0-50+j*20);
                //linia poziomu
217         c=slider.getValue();
218         for (x=0; x<width; x+=1) {
219             t = (double) x /(double) width;
220             y = offset - A * Math.sin (Math.PI * f* t+Math.PI/(120)*i*f-
                Math.PI*(f-1)/2);
221             yV = offset - A * Math.cos(Math.PI * f* t+Math.PI/(120)*i*f-
                Math.PI*(f-1)/2)*c/3;
222             g.setColor(Color.red);
223             g.drawLine (310+x0, 30+(int) y0, 310+x, 30+(int) y);
224             g.setColor(new Color(0,100,0));
225             g.drawLine (310+x0, 30+(int) yV0, 310+x, 30+(int) yV);
226             x0 = x;
227             y0 = y;
228             yV0=yV;
229         }
230
```

```
231        //ekranik
232        g.setColor(new Color(20,20,20));
233        g.fillRoundRect(355,170, 110,40,2,2);
234        g.setColor(Color.white);
235        g.fillRoundRect(360,175, 100,30,2,2);
236        g.setColor(Color.black);
237        g.drawString("sin(a)", 365, 195);
238        g.drawString("V", 425, 195);
239        g.setColor(Color.red);
240        g.fillOval(400, 185, 10,10);
241        g.setColor(Color.green);
242        g.fillOval(440, 185, 10,10);
243        g.setFont(new Font("Comic Sans MS",Font.PLAIN,10));
244        g.drawString("a - kąt między prostopadłą", 400, 232);
245        g.drawString("do powierzchni ramki,", 400, 242);
246        g.drawString("a liniami pola magnet.", 400, 252);
247        g.drawString("V - indukowana SEM", 400, 265);
248    }
```

Listing 32. Metoda paint().

Ze względu na objętość kodu opiszę polecenia blokowo. Większość używanych metod powinna być już znana czytelnikowi z wcześniej omawianych apletów.

Polecenia 83-90:

Tworzą nowe zmienne oraz je inicjalizują odpowiednimi wartościami.

Polecenie 91:

Powoduje wykonanie nie przeciążonej metody „paint()” na aplecie.

Polecenia 92-93:

Podpisują potencjometr tekstem z ustaloną czcionką.

Polecenia 95-118:

Tworzą oraz modyfikują tablice współrzędnych ramki w zależności od zmieniających się wartości zmiennych sterujących.

Polecenia 120-139:

Rysują bieguny magnesu i je odpowiednio podpisują.

Polecenia 142-149:

Rysują jasne przewody od ramki do oscyloskopu oraz gniazda w oscyloskopie.

Polecenia 152-170:

Rysują linie pola znajdujące się w oddali.

Polecenia 172-175:

Rysują ramkę na podstawie wcześniej ustalonym tablic oraz klocek podłączenia ramki do przewodów.

Polecenia 178-196:

Rysują linie pola znajdując się na pierwszym planie.

Polecenia 198-207:

Rysują oscyloskop.

Polecenia 210-216:

Rysują siatkę na ekranie oscyloskopu.

Polecenia 217-229:

Na podstawie funkcji matematycznych na sinus i cosinus oraz odpowiednich zmiennych rysowane są wykresy sin(a) oraz SEM.

Polecenia 232-247:

Rysuję mały ekranik na oscyloskopie oraz krótką legendę znajdującą się pod urządzeniem.

5.7. Elektryzowanie przez tarcie.

Aplet ten jest kolejnym apletem wykorzystującym podwójne buforowanie. Jednak tym razem operacja podwójnego buforowania jest zdefiniowana w osobnej klasie „BufferedApplet” (Listing 33.), która jest rozszerzeniem (dziedziczy po klasie) klasy „Applet”. Ponieważ ciało tej klasy nie jest obszerne, pozwolę sobie zamieścić je poniżej.

```
13  public class BufferedApplet extends Applet{
14          protected boolean podwojne_buforowanie=true;
15          private Image bufor=null;
16          public BufferedApplet()
17          {
18                  super(); //wywolanie konstruktora klasy Applet
19          }
20          @Override
21          public void update(Graphics g)
22          {
23                  Rectangle dim=this.getBounds();
24                  int w=dim.width, h=dim.height;
25                  if (podwojne_buforowanie)
26                  {
27                          if (bufor==null) bufor=this.createImage(w,h);
28                          Graphics bg=bufor.getGraphics();
29                          bg.clearRect(0,0,w,h);
30                          bg.setClip(0,0,w,h);
31                          this.paint(bg);
32                          g.drawImage(bufor,0,0,null);
33                          bg.dispose();
34                  }
35                  else
36                  {
37                          g.clearRect(0,0,w,h);
38                          paint(g);
39                  }
```

```
40        }
41  }
```

Listing 33. Klasa BufferedApplet z podwójnym buforowaniem.

O dziedziczeniu jednak nie będę tutaj pisać, ponieważ omówiłem już to przy okazji omawiania języka Java (Rozdział 2.1). Konstruktor tej klasy jedyne co robi, to wywołuje konstruktor klasy, po której dziedziczy, czyli konstruktor klasy „Applet”.

Oprócz konstruktora klasa ta zawiera jeszcze definicję jednej metody, a dokładnie przeciążona zostaje metoda „update()”, której zastosowanie omawiałem już przy okazji pierwszego apletu. Ciało tej metody jednak opiszę dokładnie.

W pierwszym wierszu (wiersz 23) tworzony jest obiekt typu „Rectangle”, czyli prostokąt. Metoda „getBounds()” wywołana na „this”, czyli na aplecie powoduje, że zostaje utworzony obiekt o wymiarach i pozycji apletu. W kolejnej linijce (wiersz 24) pobierane są wymiary tego prostokąta do odpowiednich zmiennych „w” i „h”. Warunek w wierszu 25 daje nam możliwość wyłączenia podwójnego buforowania przez ustawienie zmiennej „podwojne_buforowanie” na wartość „false”. Wtedy funkcja „update()” wyczyści tylko obszar rysowania (wiersz 37) i wywoła funkcję „paint()” z parametrem w postaci bieżącego kontekstu graficznego „g”. Jeśli jednak podwójne buforowanie jest włączone, to instrukcje są bardziej złożone. W rzeczy samej ciąg tych instrukcji jest tym samym buforowaniem, które było stosowane w czterech pierwszych apletach. Dla przypomnienia i ze względu na to, że buforowanie stanowi tu odrębną klasę omówię te instrukcje jeszcze raz. Nie będę jednak tak szczegółowo opisywać już używanych metod. W wierszu 27 sprawdzane jest czy obiekt „bufor” typu „Image” jest „null” (pusty). Jeśli tak, to tworzony jest nowy obraz typu „Image” o wymiarach apletu i przypisywany do obiektu „bufor”. W kolejnej linijce (wiersz 28) tworzony jest kontekst graficzny w ramach obiektu „bufor”. Następnie (wiersz 29) nowy kontekst graficzny jest czyszczony. Kolejna linijka kodu (wiersz 30) jest opcjonalna i usunięcie jej nic by nie zmieniło w działaniu klasy. Metoda ta po prostu wykrawa z kontekstu obszar o wymiarach określonych przez parametry. Jako, że obszar ten stanowi obszar całego kontekstu, użycie metody „setClip()” nie jest tu niezbędne. Funkcja ta przydaje się jednak, jakbyśmy chcieli rysować tylko po wybranej, mniejszej części apletu i dlatego pozostawią ją tu jako ciekawostkę.

Wiersz 31 jest wywołaniem funkcji „paint()” (najczęściej przeciążonej i zdefiniowanej przez programistę) z nowym kontekstem graficznym obiektu „bufor”. Po odmalowaniu wszystkich komponentów na tymczasowym kontekście, gotowy obraz jest

odmalowywany na bieżącym (właściwym) kontekście, jakim jest nasz aplet. W linijce 33 usuwany jest tymczasowy kontekst i zwalniane są zasoby systemowe używane przez niego. I to by było na tyle w kwestii buforowania. Powyższe wyjaśnienia będą również stanowiły źródło wiedzy dla kolejnych dwóch omawianych apletów, gdyż użyłem w nich identycznej klasy buforującej.

Teraz zajmę się właściwą klasą tworzącą aplet. Zaczną od metody „initComponents()". Jej ciała nie będę tu przedstawiał, gdyż jest to metoda tworzona automatycznie przez środowisko. W jej ciele odbywa się załadowanie elementów dodawanych w widoku projektu, jak np. przyciski czy pola tekstowe czy ustawianie rozmiarów elementów i samego apletu. Zazwyczaj jest pierwszą metodą wywoływaną w ciele metody „init()" (Listing 34.).

```
29  public void init() {
30          try {
31              java.awt.EventQueue.invokeAndWait(new Runnable() {
32                  public void run() {
33                      initComponents();
34                      laduj();
35                      setBackground(new Color(0,150,0));
36                  }
37              });
38          } catch (Exception ex) {
39              ex.printStackTrace();
40          }
41  }
```

Listing 34. Metoda init().

Ta metoda z kolei uruchamia się jako pierwsza przy ładowaniu apletu, o czym wspominałem wcześniej przy omawianiu języka Java oraz w rozdziale na temat pierwszego apletu. W dalszej kolejności wywołana jest metoda „laduj()", która ładuje obrazki (linijka, szmatka, plus, minus) z dysku do odpowiednich zmiennych oraz ustawiane jest tło apletu. Metoda „init()" wygląda tu nieco bardziej skomplikowanie niż w poprzednich apletach, lecz jedyne co się tutaj zmieniło, to fakt, że wszystkie

wykonywane w niej metody są umieszczone w osobnym wątku, a o wątkach można poczytać w rozdziale o języku Java.

Kolejne metody zdefiniowane w programie oprócz metody „paint()” są już ciągiem metod wywołanych jedynie przez użytkownika, o czym za chwilę będzie można się przekonać. Użytkownik naciskając przycisk „START” znajdujący się na aplecie powoduje wykonanie metody „jButton1ActionPerformed()” (Listing 35.), która jest metodą wykonywaną automatycznie po naciśnięciu przypisanego do niej przycisku.

```
260     private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
261         i=0; a=0; b=0;c=0;p=0;d=true;
262         jButton1.setEnabled(false);
263         this.repaint();
264         Uruchom();
265     }
```

Listing 35. Metoda obsługująca przycisk START.

W wierszu 260 ustawiane są odpowiednie zmienne sterujące ruchem komponentów na aplecie. Dalej następuje zablokowanie przycisku i odmalowanie apletu poprzez wywołanie funkcji „repaint()”. Ostatecznie wywoływana jest metoda „Uruchom()”, która rozpoczyna cykl następujących po sobie metod. Aby zrozumieć, jak te metody sterują działaniem apletu, najpierw trzeba jednak zaznajomić się z metodą „paint()” (Listing 36.), która odmalowuje komponenty na aplecie.

```
58  public void paint(Graphics g)
59  {
60          super.paint(g);
61          //linijka
62          g.drawImage(img1, A, A+10,this);
63          //plusy i minusy (elektrony i protony) na linijce
64          g.drawImage(minus, A+40, A+170, this);
65          g.drawImage(minus, A+120, A+110, this);
66          g.drawImage(minus, A+200, A+50, this);
67          g.drawImage(minus, A+70, A+115, this);
68          g.drawImage(minus, A+150, A+60, this);
```

```
69      g.drawImage(plus, A+90, A+130, this);
70      g.drawImage(plus, A+160, A+80, this);
71      g.drawImage(plus, A+30, A+140, this);
72      g.drawImage(plus, A+110, A+85, this);
73      g.drawImage(plus, A+190, A+30, this);
74      if(d)
75      {
76          //futerko
77          g.drawImage(img2, A+200+b-i, A+80-b, this);
78          //plusy i minusy (elektrony i protony) na futerku
79          if (a<2)
80          {
81              g.drawImage(minus, A+215+b-i, A+145-b, this); // minus 1
82              g.drawImage(minus, A+265+b-i, A+100-b, this); // minus 2
83          } else
84          {
85              g.setColor(Color.yellow);
86              g.fillOval(A+212-i, A+137, 22, 22); // minus 1
87              g.fillOval(A+262-i+c, A+92-c, 22, 22); // minus2
88              g.drawImage(minus, A+215-i, A+145, this);  // minus 1
89              g.drawImage(minus, A+265-i+c, A+100-c, this); // minus 2
90          }
91          //na stałe na szmatce
92          g.drawImage(plus, A+235+b-i, A+145-b, this); // plus 1
93          g.drawImage(plus, A+285+b-i, A+105-b, this); //plus 2
94          g.drawImage(plus, A+240+b-i, A+120-b, this); //plus 3
95          g.drawImage(minus, A+260+b-i, A+130-b, this); //minus 3
96      }else
97      { //powrot szmatki
98          g.drawImage(img2, A+200+b-i+p, A+80-b, this); //szmatka
99          g.setColor(Color.yellow);
100         g.fillOval(A+212-i, A+137, 22, 22); //minus 1
101         g.fillOval(A+262-i+c, A+92-c, 22, 22); // minus 2
102         g.fillOval(A+230+b-i+p, A+140-b, 22, 22); // plus 1
```

```
103             g.fillOval(A+280+b-i+p, A+100-b, 22, 22); // plus 2
104
105             g.drawImage(minus, A+215-i, A+145, this);  // minus 1
106             g.drawImage(minus, A+265-i+c, A+100-c, this); // minus 2
107             g.drawImage(minus, A+260+b-i+p, A+130-b, this); //minus 3
108             g.drawImage(plus, A+235+b-i+p, A+145-b, this); // plus 1
109             g.drawImage(plus, A+285+b-i+p, A+105-b, this); // plus 2
110             g.drawImage(plus, A+240+b-i+p, A+120-b, this); //plus 3
111         }
112     }
```

Listing 36. Metoda paint().

Na samym początku (wiersz 60) tej metody wywoływana jest bazowa (nie przeciążona) metoda „paint()”, która odrysowuje podstawowe komponenty (np. przycisk). Polecenia 62-73 rysują linijkę oraz umieszczają na niej plusy i minusy. Występująca zmienna „A” umożliwia zmianę ogólnego położenia elementów na aplecie. W kolejnym wierszu (wiersz 74) spotykamy się z warunkiem, który odpowiada za rysowanie szmatki (futerka) od startu do zakończenia pocierania (wiersze 76-95) oraz rysowania szmatki podczas powrotu do pozycji pierwotnej (wiersze 98-110). W pierwszym bloku poleceń rysowana jest poruszająca się szmatka (wiersz 77) sterowana zmiennymi „b” i „i”. Dalej w zależności od spełnienia warunku najpierw rysowane są dwa minusy pozostające cały czas na szmatce (wiersze 81-82), by potem (kiedy szmatka potrze dwa razy linijkę) przejść ze szmatki na tą linijkę i podświetlić się żółtym kolorem (wiersze 85-89). W ostatnich czterech wierszach tego bloku rysowane są plusy i minusy będące stale na szmatce.

W drugim bloku (wiersze 98-110) rysowane są plusy i minus pozostające na szmatce oraz dwa minusy, które ją opuściły i pozostały na linijce. Dwa nadmiarowe plusy na szmatce i minusy na linijce są podświetlane na żółto. Oczywiście rysowana jest też sama szmatka (wiersz 98).

W kolejnym metodach zastosowane jest tworzenie wątków, które omawiałem już przy okazji pierwszego apletu, dlatego omówię je w skrócie. Metoda „Uruchom()” (Listing 37.) tworzy i uruchamia pierwszy wątek, które przesuwa szmatkę w kierunku linijki.

```
115     public void Uruchom()
116     {
```

```
117         Timer latanie = new Timer();
118         latanie.schedule(new TimerTask(){
119
120             @Override
121             public void run() {
122                 i++;
123                 if(i>150)
124                 {
125                     this.cancel();
126                     pocieranie();
127                 }
128                 repaint();
129             }
130
131         }, 0,5);
132     }
```

Listing 37. Metoda Uruchom().

Po zakończeniu przesuwania (gdy zmienna „i” większa od 150) wątek jest kończony i uruchamia się kolejna metoda „pocieranie()” (Listing 38.), która jak sama nazwa wskazuje odpowiada za pocieranie linijki szmatką.

```
134     public void pocieranie()
135     {
136         a++;
137         Timer potrzyj = new Timer();
138         potrzyj.schedule(new TimerTask(){
139
140             @Override
141             public void run() {
142                 b++;
143                 if(a==2) c+=2;
144                 if(b>=30)
145                 {
```

```
146                         this.cancel();
147                         pocieraniedol();
148                     }
149                     repaint();
150                 }
151
152         }, 0,30);
153 }
```

Listing 38. Metoda pocieranie().

Metoda ta modyfikuje również odpowiednie zmienne („a” i „c”) sterujące minusami przechodzącymi ze szmatki na linijkę. Metoda ta odpowiada za tarcie w górę, natomiast za tarcie w dół odpowiedzialna jest metoda „pocieraniedol()” (Listing 39.) wywoływana po zakończeniu pocierania w górę.

```
155 public void pocieraniedol()
156 {
157         if (a!=3)
158         {
159         Timer trzyj = new Timer();
160         trzyj.schedule(new TimerTask(){
161
162                 @Override
163                 public void run() {
164                     b--;
165                     if(b<1)
166                     {
167                         this.cancel();
168                         pocieranie();
169                     }
170                     repaint();
171                 }
172
173         }, 0,30);
```

```
174         }
175         if(a==3)
176         {
177             d=false;
178             powrot();
179         }
180     }
```

Listing 39. Metoda pocieraniedol().

Metoda ta po zakończeniu tarcie w dół wywołuje ponownie metodę „pocieranie()” i tak metody wywołują się wzajemnie kilka razy tworząc efekt pocierania szmatki o linijkę. Jak skończy się już pocieranie szmatki, to w metodzie „pocieraniedol()” wywoływana jest metoda „powrot()” (Listing 40.), która odpowiada za ruch szmatki, w kierunku pozycji pierwotnej.

```
182     public void powrot()
183     {
184         Timer powr=new Timer();
185         powr.schedule(new TimerTask(){
186
187             @Override
188             public void run() {
189                 p+=2;
190                 if (p>100)
191                 {
192                     jButton1.setEnabled(true);
193                     this.cancel();
194                 }
195                 repaint();
196             }
197
198         }, 0, 20);
199     }
```

Listing 40. Metoda powrot().

Po zakończeniu ostatniego wątku, w którym wykonywany jest powrót, zostaje odblokowany przycisk „START” i użytkownik może od nowa uruchomić symulację.

6. Podsumowanie

Uważam, że praca ta spełnia swoje zadania dydaktyczne i będzie stanowiła źródło wiedzy nie tylko dla młodego pokolenia fizyków, ale również dla osób, które chcą zgłębić dziedzinę wiedzy, jaką jest programowanie w języku Java. W pracy tej oprócz opisów zjawisk fizycznych za pomocą definicji i odpowiednich wzorów zawarłem szczegółowy opis kodów źródłowych zawartych w niej apletów. Dzięki temu praca ta stała się również małym kursem programowania. Kody opisane są w przystępny dla czytelnika opis i pozwalają zrozumieć dokładnie każdą jego linijkę.

Ważnym aspektem są również problemy napotkane w trakcie tworzenia tej pracy. Jednym z nich był spowodowany wykorzystaniem programu Microsoft Office Publisher 2003. Moim zdaniem Microsoft nie postarał się zbytnio przy tworzeniu tego oprogramowania, gdyż nie jest kompatybilne z ich własnym produktem, jakim jest Internet Explorer. Podczas wyświetlania utworzonej witryny w przeglądarce nie pojawia się menu z odnośnikami do stron. Problem ten można jednak rozwiązać rozgrupowując menu na pojedyncze elementy. Ale czy nie mogłoby to działać od razu?

Innym problem było rzutowanie przestrzeni w aplecie z obracającą się ramką. Aby poprawnie i zgodnie z wykresami wyświetlać obracającą się ramkę konieczne było zastosowanie takie rzutu, jakby obserwator siedział tuż naprzeciw ramki (widząc od wewnątrz oba bieguny magnesu). Rzut z góry na ten układ skutkowałby tym, że ciężko byłoby zsynchronizować zmieniające się wykresy z obracającą się ramką.

I na tym zakończę moją pracę.

7. Literatura

[1] Halliday D., Resnick R., Walker J., *Podstawy Fizyki 3*, Wydawnictwo Naukowe PWN, Warszawa 2003.

[2] http://dydaktyka.fizyka.umk.pl/Elektrostatyka/Elektrostatyka.html, 30 grudzień 2009.

[3] [Ev] http://www.evertiq.pl/news/1912, 30 grudzień 2009.

[4] Rumiński J., *Język JAVA podstawy programowania*, 1999.

[5] Horstmann C. S., Cornell G., *JAVA 2 Podstawy*, Wydawnictwo HELION, Gliwice 2003.

[6] Orłowski S., *Technologia Java*, Toruń, 2009. Instytut Fizyki, Uniwersytet Mikołaja Kopernika, http://www.fizyka.umk.pl/~bigman.

[7] Orłowski S., *Java*, Toruń, 2008. Instytut Fizyki, Uniwersytet Mikołaja Kopernika, http://www.fizyka.umk.pl/~bigman.

[8] http://netbeans.org/index_pl.html, 4 stycznia 2010.

[9] http://students.mimuw.edu.pl/~lm181277/page/Java/skladnia.html - strona Laboratorium Komputerowego Wydziału MIM UW w Warszawie

8. Załączniki

8.1. Załącznik 1.

Kod źródłowy *Wsuwanie i wysuwanie magnesu*:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package paczka;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import javax.swing.JApplet;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.util.Timer;
import java.util.TimerTask;
import javax.swing.JPanel;

/**
 *
 * @author Administrator
 */
public class magnes extends JApplet {

    /**
     * Initialization method that will be called after the applet is
loaded
     * into the browser.
     */
    int i=0,a=100,dyg=1,zwrot=5;
    Image img,img1;
    protected Graphics bufferGraphics;
    protected Image offscreen;
    int [] xPoints = {70,50,50,180,180,170};
    int [] yPoints = {119,119,40,40,119,119};
    int [] xStrzalka = {145,145,145,145};
    int [] yStrzalka = {43,40,40,37};
    char [] etykieta = {'A','I','S','N'};


    @Override
    public void init() {
        setSize(300,150);
        JPanel pp = new JPanel();
        pp.setPreferredSize(new Dimension(300,150));
        pp.setBackground(new Color(204,204,255));
        add(pp);
        try
        {
            img = getImage(getDocumentBase(), "spirala.gif");
            img1 = getImage(getDocumentBase(), "spirala1.gif");
        }catch(Exception ex)
```

```
        {
            System.out.println("Nie udalo sie zaladowac obrazków");
        }
        Uruchom();
    }


//*********************************************************************
**

    @Override
    public void update(Graphics g) {
        paint(g);
    }

    @Override
    public void paint(Graphics g) {
        if (bufferGraphics == null)
        {
            offscreen = createImage(getWidth(), getHeight());
            bufferGraphics = offscreen.getGraphics();
        }
        paintComponent(bufferGraphics);
        g.drawImage(offscreen, 0, 0, null);
    }

    public void paintComponent(Graphics g){
        Graphics2D g2=(Graphics2D)g;
        super.paint(g2);
        rysuj(g2);
    }


//*********************************************************************
**

    //rysowanie po powierzchni apletu
    public void rysuj(Graphics2D g){
        g.drawImage(img1, 60, 90, this);
        g.setColor(Color.blue);
        g.fill3DRect(165-i, 95, 50, 20, true);
        g.setColor(Color.red);
        g.fill3DRect(215-i, 95, 50, 20, true);

        g.setColor(Color.white);
        g.setFont(new Font("Comic Sans MS",1,14));
        g.drawChars(etykieta, 3,1,170-i, 110);
        g.drawChars(etykieta, 2,1,250-i, 110);

        g.drawImage(img, 60, 90, this);
        g.setColor(Color.gray);
        g.drawPolyline(xPoints, yPoints, 6);

        //tarcza
        g.setColor(new Color(240,240,240));
        g.fillOval(100,28, 25, 25);
        g.setColor(new Color(120,120,120));
        g.drawOval(100,28, 25, 25);
        g.fill3DRect(112, 29, 2, 4, true);
        g.drawLine(103, 32, 106, 35);
        g.drawLine(122, 32, 119, 35);
```

```
        g.setColor(Color.red);
        g.setFont(new Font("Times New Roman",1,17));
        g.drawChars(etykieta, 0,1,107,47);

        g.setColor(Color.black);
        g.setFont(new Font("Comic Sans MS",1,12));
        g.drawChars(etykieta, 1,1,142,35);

        g.setColor(new Color(120,120,120));
        if (a!=0)
            g.drawLine(112,45,106+dyg,35-dyg); //wskazówka lewo
        else
            g.drawLine(112,45,120-dyg,35+dyg); //wskazówka prawo
        xStrzalka[1]=145+zwrot;
        xStrzalka[2]=145+zwrot;
        g.drawPolyline(xStrzalka, yStrzalka, 4); //rysowanie strzałki
    }

    //uruchamianie wątku odpowiedzialnego za wywołanie rysowania
    public void Uruchom()
    {
        Timer latanie = new Timer();
        latanie.schedule(new TimerTask(){

            @Override
            public void run() {
                if(i<a)
                {
                    a=50;
                    i++;
                    zwrot=-5;
                }
                else
                {
                    a=0;
                    i--;
                    zwrot=5;
                }
                dyg=-dyg;
                repaint();
            }
        }, 0,40);
    }
}
```

8.2. Załącznik 2.

Kod źródłowy *Wsuwanie i wysuwanie elektromagnesu*:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package paczka;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
```

```
import java.awt.Graphics2D;
import java.awt.Image;
import java.util.Timer;
import java.util.TimerTask;
import javax.swing.JApplet;
import javax.swing.JPanel;

/**
 *
 * @author Administrator
 */
public class magnes1 extends JApplet {

    int i=0,a=100,dyg=1,zwrot=5;
    Image img,img1;
    protected Graphics bufferGraphics;
    protected Image offscreen;
    int [] xPoints = {70,50,50,180,180,170};
    int [] yPoints = {119,119,40,40,119,119};
    int [] xStrzalka = {145,145,145,145};
    int [] yStrzalka = {43,40,40,37};
    int [] xObwod1 = {265,275,275,210,210,245};
    int [] yObwod1 = {105,105,70,70,20,20};
    int [] xObwod2 = {250,290,290,280,280,265};
    int [] yObwod2 = {20,20,70,70,110,110};
    char [] etykieta = {'A','I','U'};


    @Override
    public void init() {
        setSize(300,150);
        JPanel pp = new JPanel();
        pp.setPreferredSize(new Dimension(300,150));
        pp.setBackground(new Color(204,204,255));
        add(pp);
        try
        {
        img = getImage(getDocumentBase(), "spirala.gif");
        img1 = getImage(getDocumentBase(), "spirala1.gif");
        }catch(Exception ex)
        {
            System.out.println("Nie udalo sie zaladowac obrazków");
        }
        Uruchom();
    }


//*********************************************************************
**

    @Override
    public void update(Graphics g) {
        paint(g);
    }

    @Override
    public void paint(Graphics g) {
        setBackground(new Color(204, 204, 255));
        if (bufferGraphics == null)
        {
            offscreen = createImage(getWidth(), getHeight());
```

```
            bufferGraphics = offscreen.getGraphics();
        }
            paintComponent(bufferGraphics);
            g.drawImage(offscreen, 0, 0, null);
    }

    public void paintComponent(Graphics g){
        Graphics2D g2=(Graphics2D)g;
        super.paint(g2);
        rysuj(g2);
    }


//*********************************************************************
**

    //rysowanie po powierzchni apletu
    public void rysuj(Graphics2D g){
        //rysowanie magnesu
        g.drawImage(img1, 60, 90, this);
        g.setColor(Color.gray);
        g.fill3DRect(165-i, 95, 100, 20, true);

        g.setColor(Color.red);
        for (int j=0;j<9;j++)
        {
            g.drawLine(173-i+j*10, 115,178-i+j*10, 95);
            g.drawArc(178-i+j*10, 93, 3, 3, 0, 180);
            g.drawArc(170-i+j*10, 113, 3, 3, 0, -180);
        }

        //obwód
        g.drawImage(img, 60, 90, this);
        g.setColor(Color.gray);
        g.drawPolyline(xPoints, yPoints, 6);

        //obwód elektromagnesu
        g.setColor(new Color(240,240,240));
        xObwod1[0]=xObwod2[5]=265-i;
        g.drawImage(img, 60, 90, this);
        g.setColor(Color.red);
        g.drawPolyline(xObwod1,yObwod1, 6);
        g.drawPolyline(xObwod2,yObwod2, 6);
        g.drawLine(245, 10, 245, 30);
        g.drawLine(250, 15, 250, 25);
        g.drawChars(etykieta, 2, 1, 249, 13);

        //tarcza
        g.setColor(new Color(240,240,240));
        g.fillOval(100,28, 25, 25);
        g.setColor(new Color(120,120,120));
        g.drawOval(100,28, 25, 25);
        g.fill3DRect(112, 29, 2, 4, true);
        g.drawLine(103, 32, 106, 35);
        g.drawLine(122, 32, 119, 35);

        g.setColor(Color.red);
        g.setFont(new Font("Times New Roman",1,17));
        g.drawChars(etykieta, 0,1,107,47);

        g.setColor(Color.black);
```

```
        g.setFont(new Font("Comic Sans MS",1,12));
        g.drawChars(etykieta, 1,1,142,35);

        g.setColor(new Color(120,120,120));
        if (a!=0)
            g.drawLine(112,45,106+dyg,35-dyg); //wskazówka lewo
        else
            g.drawLine(112,45,120-dyg,35+dyg); //wskazówka prawo
        xStrzalka[1]=145+zwrot;
        xStrzalka[2]=145+zwrot;
        g.drawPolyline(xStrzalka, yStrzalka, 4); //rysowanie strzałki
    }

    //uruchamianie wątku odpowiedzialnego za wywołanie rysowania
    public void Uruchom()
    {
        Timer latanie = new Timer();
        latanie.schedule(new TimerTask(){

            @Override
            public void run() {
                if(i<a)
                {
                    a=50;
                    i++;
                    zwrot=-5;
                }
                else
                {
                    a=0;
                    i--;
                    zwrot=5;
                }
                dyg=-dyg;
                repaint();
            }

        }, 0,40);
    }
}
```

8.3. Załącznik 3.

Kod źródłowy *Włączanie i wyłączanie prądu w zwojnicy elektromagnesu*:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package paczka;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.util.Timer;
import java.util.TimerTask;
import javax.swing.JApplet;
```

```
import javax.swing.JPanel;

/**
 *
 * @author Administrator
 */
public class magnes2 extends JApplet {

    int i=0,a=100,dyg=1,zwrot=5;
    Image img,img1;
    protected Graphics bufferGraphics;
    protected Image offscreen;
    int [] xPoints = {70,50,50,180,180,170};
    int [] yPoints = {119,119,40,40,119,119};
    int [] xStrzalka = {145,145,145,145};
    int [] yStrzalka = {43,40,40,37};
    int [] xObwod1 = {165,225,225,210,210,225};
    int [] yObwod1 = {105,105,70,70,20,20};
    int [] xObwod2 = {270,290,290,230,230,165};
    int [] yObwod2 = {20,20,70,70,110,110};
    char [] etykieta = {'A','I','U'};

    @Override
    public void init() {
        setSize(300,150);
        JPanel pp = new JPanel();
        pp.setPreferredSize(new Dimension(300,150));
        pp.setBackground(new Color(204,204,255));
        add(pp);
        try
        {
        img = getImage(getDocumentBase(), "spirala.gif");
        img1 = getImage(getDocumentBase(), "spirala1.gif");
        }catch(Exception ex)
        {
            System.out.println("Nie udalo sie zaladowac obrazków");
        }
        Uruchom();
    }


//*********************************************************************
**
    //  metoda wywołujące buforowane odrysowywanie apletu
    @Override
    public void update(Graphics g) {
        paint(g);
    }

    @Override
    public void paint(Graphics g) {
        setBackground(new Color(204, 204, 255));
        if (bufferGraphics == null)
        {
            offscreen = createImage(getWidth(), getHeight());
            bufferGraphics = offscreen.getGraphics();
        }
            paintComponent(bufferGraphics);
            g.drawImage(offscreen, 0, 0, null);
    }
```

```
    public void paintComponent(Graphics g){
        Graphics2D g2=(Graphics2D)g;
        super.paint(g2);
        rysuj(g2);
    }


//*********************************************************************
**

    //rysowanie po powierzchni apletu
    public void rysuj(Graphics2D g){
        //rysowanie magnesu
        g.drawImage(img1, 60, 90, this);
        g.setColor(Color.LIGHT_GRAY);
        g.fill3DRect(65, 95, 100, 20, true);

        g.setColor(Color.red);
        for (int j=0;j<9;j++)
        {
            g.drawLine(70+j*10, 115,78+j*10, 95);
            g.drawArc(78+j*10, 93, 3, 3, 0, 180);
            g.drawArc(70+j*10, 113, 3, 3, 0, -180);
        }
        //obwód
        g.drawImage(img, 60, 90, this);
        g.setColor(Color.gray);
        g.drawPolyline(xPoints, yPoints, 6);

        //obwód elektromagnesu
        g.setColor(new Color(240,240,240));
        g.drawImage(img, 60, 90, this);
        g.setColor(Color.red);
        g.drawPolyline(xObwod1,yObwod1, 6);
        g.drawPolyline(xObwod2,yObwod2, 6);
        g.drawLine(225, 10, 225, 30);
        g.drawLine(230, 15, 230, 25);
        g.drawChars(etykieta, 2, 1, 229, 13);
        g.drawLine(230, 20, 250, 20);
        g.drawOval(250, 17,3,3);
        g.drawOval(270, 17,3,3);
        g.drawLine(250,17,268-zwrot,12-zwrot); //przełącznik

        //tarcza
        g.setColor(new Color(240,240,240));
        g.fillOval(100,28, 25, 25);
        g.setColor(new Color(120,120,120));
        g.drawOval(100,28, 25, 25);
        g.fill3DRect(112, 29, 2, 4, true);
        g.drawLine(103, 32, 106, 35);
        g.drawLine(122, 32, 119, 35);

        g.setColor(Color.red);
        g.setFont(new Font("Times New Roman",1,17));
        g.drawChars(etykieta, 0,1,107,47);

        g.setFont(new Font("Comic Sans MS",1,12));
        g.setColor(Color.black);
        if ((a==50)&&(i<=8))
        {
            g.drawLine(112,45,120-dyg,35+dyg); //wskazówka prawo
```

```
            g.drawChars(etykieta, 1,1,142,36);
            g.setColor(new Color(120,120,120));
            xStrzalka[1]=145+zwrot;
            xStrzalka[2]=145+zwrot;
            g.drawPolyline(xStrzalka, yStrzalka, 4); //rysowanie strzałki
        }
        else
            if ((a==0)&&(i>=42))
            {
                g.drawLine(112,45,106+dyg,35-dyg); //wskazówka lewo
                g.drawChars(etykieta, 1,1,142,36);
                g.setColor(new Color(120,120,120));
                xStrzalka[1]=145+zwrot;
                xStrzalka[2]=145+zwrot;
                g.drawPolyline(xStrzalka, yStrzalka, 4); //rysowanie
strzałki
            }
            else
            {
                g.drawLine(112,45,112,35);
                g.drawLine(113,45,113,35);
            } //wskazówka zero
    }

    //uruchamianie wątku odpowiedzialnego za wywołanie rysowania
    public void Uruchom()
    {
        Timer latanie = new Timer();
        latanie.schedule(new TimerTask(){

            @Override
            public void run() {
                if(i<a)
                {
                    a=50;
                    i++;
                    zwrot=5;
                }
                else
                {
                    a=0;
                    i--;
                    zwrot=-5;
                }
                dyg=-dyg;
                repaint();
            }

        }, 0,50);
    }
}
```

8.4. Załącznik 4.

Kod źródłowy *Elektromagnes z opornicą suwakową*:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */
```

```
package paczka;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.util.Timer;
import java.util.TimerTask;
import javax.swing.JApplet;
import javax.swing.JPanel;

/**
 *
 * @author Administrator
 */
public class magnes3 extends JApplet {

    int i=0,a=100,dyg=1,zwrot=5;
    Image img,img1;
    protected Graphics bufferGraphics;
    protected Image offscreen;
    int [] xPoints = {70,50,50,180,180,170};
    int [] yPoints = {119,119,40,40,119,119};
    int [] xStrzalka = {145,145,145,145};
    int [] yStrzalka = {43,40,40,37};
    int [] xObwod1 = {165,225,225,210,210,225};
    int [] yObwod1 = {105,105,70,70,20,20};
    int [] xObwod2 = {270,290,290,230,230,165};
    int [] yObwod2 = {20,20,70,70,110,110};
    int [] xPot = {240,240,250,250};
    int [] yPot = {20,7,7,15};
    int [] xStrzalka1 = {247,250,253};
    int [] yStrzalka1 = {11,14,11};
    char [] etykieta = {'A','I','U'};


    @Override
    public void init() {
        setSize(300,150);
        JPanel pp = new JPanel();
        pp.setPreferredSize(new Dimension(300,150));
        pp.setBackground(new Color(204,204,255));
        add(pp);
        try
        {
        img = getImage(getDocumentBase(), "spirala.gif");
        img1 = getImage(getDocumentBase(), "spirala1.gif");
        }catch(Exception ex)
        {
            System.out.println("Nie udalo sie zaladowac obrazków");
        }
        Uruchom();
    }


//*********************************************************************
**
    //  metoda wywołujące buforowane odrysowywanie apletu
    @Override
```

```
    public void update(Graphics g) {
        paint(g);
    }

    @Override
    public void paint(Graphics g) {
        setBackground(new Color(204, 204, 255));
        if (bufferGraphics == null)
        {
            offscreen = createImage(getWidth(), getHeight());
            bufferGraphics = offscreen.getGraphics();
        }
            paintComponent(bufferGraphics);
            g.drawImage(offscreen, 0, 0, null);
    }

    public void paintComponent(Graphics g){
        Graphics2D g2=(Graphics2D)g;
        super.paint(g2);
        rysuj(g2);
    }


//*********************************************************************
**

    //rysowanie po powierzchni apletu
    public void rysuj(Graphics2D g){
        //rysowanie magnesu
        g.drawImage(img1, 60, 90, this);
        g.setColor(Color.LIGHT_GRAY);
        g.fill3DRect(65, 95, 100, 20, true);

        g.setColor(Color.red);
        for (int j=0;j<9;j++)
        {
            g.drawLine(70+j*10, 115,78+j*10, 95);
            g.drawArc(78+j*10, 93, 3, 3, 0, 180);
            g.drawArc(70+j*10, 113, 3, 3, 0, -180);
        }

        //obwód
        g.drawImage(img, 60, 90, this);
        g.setColor(Color.gray);
        g.drawPolyline(xPoints, yPoints, 6);
        g.drawImage(img, 60, 90, this);

        //obwód elektromagnesu
        g.setColor(Color.red);
        g.drawPolyline(xObwod1,yObwod1, 6);
        g.drawPolyline(xObwod2,yObwod2, 6);
        g.drawLine(225, 10, 225, 30);
        g.drawLine(230, 15, 230, 25);
        g.drawChars(etykieta, 2, 1, 229, 13);
        g.drawLine(230, 20, 240, 20);

        g.draw3DRect(250, 15, 20, 10, true); //potencjometr
        xPot[2]=xPot[3]=250+i/2;
        g.drawPolyline(xPot,yPot,4);

        xStrzalka1[0]=247+i/2;
```

```
        xStrzalka1[1]=250+i/2;
        xStrzalka1[2]=253+i/2;
        g.drawPolyline(xStrzalka1,yStrzalka1,3);

        //tarcza
        g.setColor(new Color(240,240,240));
        g.fillOval(100,28, 25, 25);
        g.setColor(new Color(120,120,120));
        g.drawOval(100,28, 25, 25);
        g.fill3DRect(112, 29, 2, 4, true);
        g.drawLine(103, 32, 106, 35);
        g.drawLine(122, 32, 119, 35);

        g.setColor(Color.red);
        g.setFont(new Font("Times New Roman",1,17));
        g.drawChars(etykieta, 0,1,107,47);

        g.setFont(new Font("Comic Sans MS",1,12));
        g.setColor(Color.black);
        if (a!=0)
            g.drawLine(112,45,106+dyg,35-dyg); //wskazówka lewo
        else
            g.drawLine(112,45,120-dyg,35+dyg); //wskazówka prawo
        xStrzalka[1]=145+zwrot;
        xStrzalka[2]=145+zwrot;
        g.drawPolyline(xStrzalka, yStrzalka, 4); //rysowanie strzałki
        g.drawChars(etykieta, 1,1,142,36);
    }

    //uruchamianie wątku odpowiedzialnego za wywołanie rysowania
    public void Uruchom()
    {
        Timer latanie = new Timer();
        latanie.schedule(new TimerTask(){

            @Override
            public void run() {
                if(i<a)
                {
                    a=40;
                    i++;
                    zwrot=-5;
                }
                else
                {
                    a=0;
                    i--;
                    zwrot=5;
                }
                dyg=-dyg;
                repaint();
            }

        }, 0,50);
    }
}
```

8.5. Załącznik 5.

Kod źródłowy *Ramka wchodząca w pole magnetyczne*:

Plik *DrawingPanel.java*:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package org.me.ramka;

import javax.swing.*;

/**
 *
 * @author KJ
 */
public class DrawingPanel extends JPanel {
        java.awt.image.BufferedImage  buf;

         /** Odrysowywanie panelu
         *@param g obszar do rysowania
         */
    @Override
      public void paintComponent (java.awt.Graphics g)
        {
            super.paintComponent(g);
            g.drawImage (buf, 0, 0, this);
      }
}
```

Plik *RamkaAplet.java*:

```
/*
 * RamkaAplet.java
 *
 * Created on 14 lipiec 2008, 13:03
 */

package org.me.ramka;

import java.awt.Color;
import java.awt.image.BufferedImage;
import java.awt.Graphics2D;
import java.math.BigDecimal;
import java.text.NumberFormat;
import java.util.Locale;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.swing.SpinnerNumberModel;


/**
 *
 * @author  KJ
 */
public class RamkaAplet extends javax.swing.JApplet {

    /** Initializes the applet RamkaAplet */
    @Override
    public void init() {
        try {
```

```
            java.awt.EventQueue.invokeAndWait(new Runnable() {
                public void run() {
                    setSize(540, 400);
                    initComponents();
                    SpinnerNumberModel modelv=new
SpinnerNumberModel(0.1,0,20,0.01);
                    SpinnerNumberModel modelB=new
SpinnerNumberModel(1.00,0.001,10,0.001);
                    SpinnerNumberModel modelx=new SpinnerNumberModel(70,
1, 200, 1);
                    SpinnerNumberModel modely=new SpinnerNumberModel(100,
1, 200, 1);
                    jSpinner1.setModel(modelx);
                    jSpinner2.setModel(modely);
                    jSpinner4.setModel(modelv);
                    jSpinner3.setModel(modelB);
                    init_boki();
                    rys_ramka();
                }
            });
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    /** This method is called from within the init() method to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {

        jButton1 = new javax.swing.JButton();
        jSpinner1 = new javax.swing.JSpinner();
        jLabel1 = new javax.swing.JLabel();
        jLabel2 = new javax.swing.JLabel();
        jSpinner2 = new javax.swing.JSpinner();
        jLabel3 = new javax.swing.JLabel();
        jSpinner3 = new javax.swing.JSpinner();
        jLabel4 = new javax.swing.JLabel();
        jLabel5 = new javax.swing.JLabel();
        jLabel6 = new javax.swing.JLabel();
        jLabel7 = new javax.swing.JLabel();
        jSpinner4 = new javax.swing.JSpinner();
        jLabel8 = new javax.swing.JLabel();
        dp1 = new org.me.ramka.DrawingPanel();
        jLabel9 = new javax.swing.JLabel();
        dp2 = new org.me.ramka.DrawingPanel();
        jLabel10 = new javax.swing.JLabel();
        jLabel14 = new javax.swing.JLabel();
        jLabel16 = new javax.swing.JLabel();
        dp3 = new org.me.ramka.DrawingPanel();
        jLabel11 = new javax.swing.JLabel();
        jLabel17 = new javax.swing.JLabel();
        jLabel15 = new javax.swing.JLabel();
        jButton2 = new javax.swing.JButton();
        jLabel12 = new javax.swing.JLabel();
        jLabel13 = new javax.swing.JLabel();
```

```
        setBackground(java.awt.SystemColor.control);

        jButton1.setText("Start");
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });

        jSpinner1.setValue(70);
        jSpinner1.addChangeListener(new
javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                x_change(evt);
            }
        });

        jLabel1.setText("Bok x:");

        jLabel2.setText("Bok y:");

        jSpinner2.setValue(100);
        jSpinner2.addChangeListener(new
javax.swing.event.ChangeListener() {
            public void stateChanged(javax.swing.event.ChangeEvent evt) {
                y_change(evt);
            }
        });

        jLabel3.setText(" B =");

        jSpinner3.setValue(1);

        jLabel4.setText("cm");

        jLabel5.setText("cm");

        jLabel6.setText("T");

        jLabel7.setText(" v =");

        jSpinner4.setValue(10);

        jLabel8.setText("m/s");

        dp1.setBackground(new java.awt.Color(255, 255, 255));
        dp1.setMaximumSize(new java.awt.Dimension(200, 266));
        dp1.setMinimumSize(new java.awt.Dimension(200, 266));
        dp1.setPreferredSize(new java.awt.Dimension(200, 266));

        jLabel9.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/org/me/ramka/xxx.png")));
// NOI18N

        javax.swing.GroupLayout dp1Layout = new
javax.swing.GroupLayout(dp1);
        dp1.setLayout(dp1Layout);
        dp1Layout.setHorizontalGroup(

dp1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
dp1Layout.createSequentialGroup()
                .addContainerGap(63, Short.MAX_VALUE)
                .addComponent(jLabel9,
javax.swing.GroupLayout.PREFERRED_SIZE, 137,
javax.swing.GroupLayout.PREFERRED_SIZE))
        );
        dp1Layout.setVerticalGroup(

dp1Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(dp1Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel9)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
        );

        dp2.setBackground(new java.awt.Color(255, 255, 255));
        dp2.setMaximumSize(new java.awt.Dimension(299, 126));
        dp2.setMinimumSize(new java.awt.Dimension(299, 126));

        jLabel10.setText("0 Wb");
        jLabel10.setVerticalAlignment(javax.swing.SwingConstants.BOTTOM);
        jLabel10.setAlignmentY(0.0F);

        jLabel14.setFont(new java.awt.Font("Tahoma", 0, 10));

jLabel14.setHorizontalAlignment(javax.swing.SwingConstants.TRAILING);
        jLabel14.setText("Czas [s]");

        jLabel16.setFont(new java.awt.Font("Tahoma", 0, 10));
        jLabel16.setText("Strumień [Wb]");
        jLabel16.setAlignmentY(0.0F);

        javax.swing.GroupLayout dp2Layout = new
javax.swing.GroupLayout(dp2);
        dp2.setLayout(dp2Layout);
        dp2Layout.setHorizontalGroup(

dp2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(dp2Layout.createSequentialGroup()
                .addContainerGap()

.addGroup(dp2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.LEADING)
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
dp2Layout.createSequentialGroup()
                        .addComponent(jLabel16,
javax.swing.GroupLayout.DEFAULT_SIZE, 106, Short.MAX_VALUE)
                        .addGap(18, 18, 18)
                        .addComponent(jLabel10,
javax.swing.GroupLayout.PREFERRED_SIZE, 120,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(45, 45, 45))
                    .addComponent(jLabel14,
javax.swing.GroupLayout.Alignment.TRAILING)))
        );
        dp2Layout.setVerticalGroup(

dp2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(dp2Layout.createSequentialGroup()
```

```
.addGroup(dp2Layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.BASELINE)
                    .addComponent(jLabel16,
javax.swing.GroupLayout.PREFERRED_SIZE, 24,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel10))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 108,
Short.MAX_VALUE)
                .addComponent(jLabel14,
javax.swing.GroupLayout.PREFERRED_SIZE, 28,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        dp3.setBackground(new java.awt.Color(255, 255, 255));
        dp3.setMaximumSize(new java.awt.Dimension(299, 130));
        dp3.setMinimumSize(new java.awt.Dimension(299, 134));

        jLabel11.setText(" 0 V");

        jLabel17.setFont(new java.awt.Font("Tahoma", 0, 10));
        jLabel17.setText("SEM [V]");
        jLabel17.setAlignmentY(0.0F);

        jLabel15.setFont(new java.awt.Font("Tahoma", 0, 10));
        jLabel15.setText("Czas [s]");

        javax.swing.GroupLayout dp3Layout = new
javax.swing.GroupLayout(dp3);
        dp3.setLayout(dp3Layout);
        dp3Layout.setHorizontalGroup(

dp3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(dp3Layout.createSequentialGroup()
                .addContainerGap()
                .addComponent(jLabel17,
javax.swing.GroupLayout.PREFERRED_SIZE, 117,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(jLabel11,
javax.swing.GroupLayout.PREFERRED_SIZE, 111,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(43, Short.MAX_VALUE))
            .addGroup(dp3Layout.createSequentialGroup()
                .addContainerGap(264, Short.MAX_VALUE)
                .addComponent(jLabel15))
        );
        dp3Layout.setVerticalGroup(

dp3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(dp3Layout.createSequentialGroup()

.addGroup(dp3Layout.createParallelGroup(javax.swing.GroupLayout.Alignment
.BASELINE)
                    .addComponent(jLabel17,
javax.swing.GroupLayout.PREFERRED_SIZE, 22,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jLabel11))
```

```
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED, 91,
Short.MAX_VALUE)
                .addComponent(jLabel15,
javax.swing.GroupLayout.PREFERRED_SIZE, 24,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );

        jButton2.setFont(new java.awt.Font("Tahoma", 0, 8));
        jButton2.setText("STOP");
        jButton2.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton2ActionPerformed(evt);
            }
        });

        jLabel12.setFont(new java.awt.Font("Arial Rounded MT Bold", 2,
11));
        jLabel12.setIcon(new
javax.swing.ImageIcon(getClass().getResource("/org/me/ramka/x.png"))); //
NOI18N
        jLabel12.setText(" zwrot linii pola magnetycznego");

        jLabel13.setFont(new java.awt.Font("Arial Rounded MT Bold", 2,
10));
        jLabel13.setText("(linie skierowane za panel)");

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(getContentPane());
        getContentPane().setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                    .addGroup(layout.createSequentialGroup()
                        .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                            .addComponent(dp1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TR
AILING)
                                .addComponent(jLabel13)
                                .addComponent(jLabel12)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TR
AILING, false)
                            .addComponent(dp2,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
```

```
                            .addComponent(dp3,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)))
                    .addGroup(layout.createSequentialGroup()
                        .addGap(25, 25, 25)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TR
AILING)

.addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                                .addComponent(jLabel2)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jSpinner2,
javax.swing.GroupLayout.PREFERRED_SIZE, 70,
javax.swing.GroupLayout.PREFERRED_SIZE))

.addGroup(javax.swing.GroupLayout.Alignment.LEADING,
layout.createSequentialGroup()
                                .addComponent(jLabel1)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jSpinner1,
javax.swing.GroupLayout.PREFERRED_SIZE, 70,
javax.swing.GroupLayout.PREFERRED_SIZE)))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                            .addComponent(jLabel4)
                            .addComponent(jLabel5))
                        .addGap(14, 14, 14)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING, false)
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel7)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jSpinner4,
javax.swing.GroupLayout.PREFERRED_SIZE, 79,
javax.swing.GroupLayout.PREFERRED_SIZE))
                            .addGroup(layout.createSequentialGroup()
                                .addComponent(jLabel3)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                                .addComponent(jSpinner3,
javax.swing.GroupLayout.PREFERRED_SIZE, 79,
javax.swing.GroupLayout.PREFERRED_SIZE)))
                        .addGap(18, 18, 18)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                            .addComponent(jLabel6,
javax.swing.GroupLayout.PREFERRED_SIZE, 16,
javax.swing.GroupLayout.PREFERRED_SIZE)
                            .addComponent(jLabel8))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
                        .addComponent(jButton2,
javax.swing.GroupLayout.PREFERRED_SIZE, 59,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jButton1)))
                .addContainerGap(12, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                    .addGroup(layout.createSequentialGroup()
                        .addComponent(dp1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                        .addComponent(jLabel12)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(jLabel13)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED))
                    .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                        .addComponent(dp2,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                        .addComponent(dp3,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addGap(8, 8, 8)))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TR
AILING)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LE
ADING)
                        .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                                .addComponent(jLabel3)
                                .addComponent(jSpinner3,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jLabel6))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                                .addComponent(jLabel7)
                                .addComponent(jSpinner4,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jLabel8)))
                        .addGroup(layout.createSequentialGroup()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                                .addComponent(jLabel1)
                                .addComponent(jSpinner1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jLabel4))

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                                .addComponent(jSpinner2,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)
                                .addComponent(jLabel2)
                                .addComponent(jLabel5))))

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BA
SELINE)
                        .addComponent(jButton2,
javax.swing.GroupLayout.PREFERRED_SIZE, 34,
javax.swing.GroupLayout.PREFERRED_SIZE)
                        .addComponent(jButton1,
javax.swing.GroupLayout.DEFAULT_SIZE, 46, Short.MAX_VALUE)))
                .addContainerGap())
        );
    }// </editor-fold>//GEN-END:initComponents


public void init_boki()
{
    bokx=Double.valueOf(jSpinner1.getValue().toString());
    boky=Double.valueOf(jSpinner2.getValue().toString());
    intbokx = Integer.valueOf(jSpinner1.getValue().toString());
    intboky = Integer.valueOf(jSpinner2.getValue().toString());
}

private void init_dp1()
{
    dp1.buf = new BufferedImage(250, 250, BufferedImage.TYPE_4BYTE_ABGR);
}

private void init_dp2()
{
    dp2h = dp2.getHeight();
    dp2w = dp2.getWidth();
```

```
    dp2.buf = new BufferedImage(dp2w, dp2h,
BufferedImage.TYPE_4BYTE_ABGR);
    Graphics2D g = dp2.buf.createGraphics();
    g.setColor(Color.BLACK);
    g.drawLine(0, dp2h-1, dp2w, dp2h-1);
    g.drawLine(0, 0, 0, dp2h-1);
    //skala
    g.drawLine(35,dp2h,35,dp2h-4);
    g.drawLine(70,dp2h,70,dp2h-4);
    g.drawLine(105,dp2h,105,dp2h-4);
    g.drawLine(140,dp2h,140,dp2h-4);
    g.drawLine(175,dp2h,175,dp2h-4);
    g.drawLine(210,dp2h,210,dp2h-4);
    g.drawLine(245,dp2h,245,dp2h-4);
    g.drawLine(280,dp2h,280,dp2h-4);
    g.drawLine(1,(int)(0.25*dp2h),4,(int)(0.25*dp2h));
    g.drawLine(0,(int)((0.75/2+0.25)*dp2h),4,(int)((0.75/2+0.25)*dp2h));

    //opis skali
    czas=(float)(bokx/V);
    NumberFormat nf = NumberFormat.getInstance(Locale.ENGLISH);
    nf.setMaximumFractionDigits(1);
    nf.setMinimumFractionDigits(1);
    g.drawString(nf.format(czas), 65,dp2h-7);
    czas=czas/2;
    g.drawString(nf.format(czas), 30,dp2h-7);
    g.drawString(nf.format(3*czas), 100,dp2h-7);
    g.drawString(nf.format(4*czas), 135,dp2h-7);
    g.drawString(nf.format(5*czas), 170,dp2h-7);
    g.drawString(nf.format(6*czas), 205,dp2h-7);
    g.drawString(nf.format(7*czas), 240,dp2h-7);

    g.drawString(""+wartB,5,(int)(0.24*dp2h));
    g.drawString(""+wartB/2,5,(int)(0.61*dp2h));
    g.setColor(Color.BLUE);
    dp2.repaint();
}

private void init_dp3()
{
    dp3h = dp3.getHeight();
    dp3w = dp3.getWidth();
    dp3.buf = new BufferedImage(dp3w, dp3h,
BufferedImage.TYPE_4BYTE_ABGR);

    Graphics2D g = dp3.buf.createGraphics();
    g.setColor(Color.BLACK);

    g.drawLine(0, dp3h-1, dp3.getWidth(), dp3h-1);
    g.drawLine(0, 0, 0, dp3h-1);
    //skala
    g.drawLine(35,dp3h,35,dp3h-4);
    g.drawLine(70,dp3h,70,dp3h-4);
    g.drawLine(105,dp3h,105,dp3h-4);
    g.drawLine(140,dp3h,140,dp3h-4);
    g.drawLine(175,dp3h,175,dp3h-4);
    g.drawLine(210,dp3h,210,dp3h-4);
    g.drawLine(245,dp3h,245,dp3h-4);
    g.drawLine(280,dp3h,280,dp3h-4);

    g.drawLine(0,(int)(0.25*dp3h),4,(int)(0.25*dp3h));
```

```
    g.drawLine(0,(int)((0.75/2+0.25)*dp3h),4,(int)((0.75/2+0.25)*dp3h));

    //opis skali
    czas=(float)(bokx/V);
    NumberFormat nf = NumberFormat.getInstance(Locale.ENGLISH);
    nf.setMaximumFractionDigits(1);
    nf.setMinimumFractionDigits(1);
    g.drawString(nf.format(czas), 65,dp3h-7);
    czas=czas/2;
    g.drawString(nf.format(czas), 30,dp3h-7);
    g.drawString(nf.format(3*czas), 100,dp3h-7);
    g.drawString(nf.format(4*czas), 135,dp3h-7);
    g.drawString(nf.format(5*czas), 170,dp3h-7);
    g.drawString(nf.format(6*czas), 205,dp3h-7);
    g.drawString(nf.format(7*czas), 240,dp3h-7);

    g.drawString(""+wartSEM,5,(int)(0.24*dp3h));
    g.drawString(""+wartSEM/2,5,(int)(0.61*dp3h));
    g.setColor(Color.GREEN);
    dp3.repaint();
}

private void rys_ramka()
{
    init_dp1();
    Graphics2D g1 = dp1.buf.createGraphics();
    g1.setColor(Color.RED);
    if (intbokx>intboky)
    {
        g1.drawRect((int)((v*t)*100+(143-x*100))-65, 100, (int)(x*100),
(int)(y*50));
        g1.drawRect((int)((v*t)*100+(143-x*100))+1-65, 101, (int)(x*100)-
2, (int)(y*50)-2);
    }
    else
        if (intbokx==intboky)
        {
            g1.drawRect((int)((v*t)*100+(143-x*100))-65, 75,
(int)(x*100), (int)(x*110));
            g1.drawRect((int)((v*t)*100+(143-x*100))+1-65, 76,
(int)(x*100)-2, (int)(x*110)-2);
        }
        else
        {
            g1.drawRect((int)((v*t)*100+(143-x*100))-65, 60,
(int)(x*100), (int)(y*100));
            g1.drawRect((int)((v*t)*100+(143-x*100))+1-65, 61,
(int)(x*100)-2, (int)(y*100)-2);
        }
    dp1.repaint();
}

//ustawianie poprawnego wyświetlania wartości na osiach pionowych oby
wykresów
private void skalaY()
{
    dzielB = dzielSEM = 1;
    wartB=bokx*boky*B/10000;
    if (wartB/2>=0.001)
    {
        strB=" Wb";
```

```
    }else
        if(wartB/2>=0.000001)
        {
            wartB=wartB*1000;
            jLabel16.setText("Strumień [10^-3 Wb]");
            strB="*10^-3 Wb";
            dzielB=1000;
        }
        else
            if(wartB/2>=0.000000001)
            {
                wartB=wartB*1000000;
                jLabel16.setText("Strumień [10^-6 Wb]");
                strB="*10^-6 Wb";
                dzielB=1000000;
            }
    wartB=zaokraglanie(wartB,3);

    wartSEM=boky*B*V/10000;
    if (wartSEM/2>=0.001)
    {
        strSEM=" V";
    }else
        if (wartSEM/2>=0.000001)
        {
            wartSEM=wartSEM*1000;
            jLabel17.setText("Sem [10^-3 V]");
            strSEM="*10^-3 V";
            dzielSEM=1000;
        }
    wartSEM=zaokraglanie(wartSEM, 3);
}

//*********************************************************************
**********************

// przycisk START/PAUSE
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed
    intbokx = Integer.valueOf(jSpinner1.getValue().toString());
    intboky = Integer.valueOf(jSpinner2.getValue().toString());
    rys_ramka();
    if (start==false&&pauza==false)
    {
        ustawPoStarcie();
        skalaY(); // ustawienie poprawnych wartości na osi Y
        init_dp2();
        init_dp3();
        new Symulacja(this);
    } else if(pauza==false)
    {
        jButton1.setText("Start");
        pauza = true;
    }else
    {
        ustawPoWznowieniu();
    }
}//GEN-LAST:event_jButton1ActionPerformed

//przycisk STOP
```

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton2ActionPerformed
    ustawPoStop();
    rys_ramka();
    init_dp2();
    init_dp3();
}//GEN-LAST:event_jButton2ActionPerformed


private void ustawPoStarcie()
{
    b=c=1;
    start = true;
    pauza = false;

    bokx=Double.valueOf(jSpinner1.getValue().toString());
    boky=Double.valueOf(jSpinner2.getValue().toString());
    B=Double.valueOf(jSpinner3.getValue().toString());
    V=Double.valueOf(jSpinner4.getValue().toString())*100;

    jSpinner1.setEnabled(false);
    jSpinner2.setEnabled(false);
    jSpinner3.setEnabled(false);
    jSpinner4.setEnabled(false);
    jButton1.setText("Pause");
}

private void ustawPoWznowieniu()
{
    b=c=1;
    jButton1.setText("Pause");
    pauza = false;
}

public void ustawPoStop()
{
    jButton1.setText("Start");
    jLabel10.setText("0 Wb");
    jLabel11.setText("0 V");
    start = false;
    pauza = false;
    jSpinner1.setEnabled(true);
    jSpinner2.setEnabled(true);
    jSpinner3.setEnabled(true);
    jSpinner4.setEnabled(true);
    t = t1 = strum1 = 0;
}

//*************************************************************************
**********************
//odrysowywanie ramki podczas zmiany szerokości lub długości ramki
private void x_change(javax.swing.event.ChangeEvent evt) {//GEN-
FIRST:event_x_change
    init_boki();
    rys_ramka();
}//GEN-LAST:event_x_change

private void y_change(javax.swing.event.ChangeEvent evt) {//GEN-
FIRST:event_y_change
    init_boki();
    rys_ramka();
```

```
}//GEN-LAST:event_y_change

//*********************************************************************
**********************

class Symulacja implements Runnable
{
      RamkaAplet parent;
      Thread active;

        Symulacja(RamkaAplet p)
      {
            parent = p; // zachowanie uchwytu do głównej klasy
            active = new Thread(this); //utworzenie nowego wątku
            active.start(); // wystartowanie wątku
      }

        public void run()
        {
            //tworzenie grafiki do rysowania po panelach
            Graphics2D g2 = dp2.buf.createGraphics();
            Graphics2D g3 = dp3.buf.createGraphics();
            g2.setColor(Color.BLUE);
            g3.setColor(new Color(50,200,50));

            strum1 = 0;
            maxStrum = B*bokx*boky/10000; //maksymalny strumień
            sem2 = B*V*boky/10000; //maksymalne SEM i stałe do czasu
wejścia całej ramki w pole
            deltat = V/100; //krok czasowy
            do
            {

                t = t + deltat; //bieżąca chwila czasu

                if(v*t<=x) //obliczanie strumienia w chwili bieżącej,
                {           //gdy ramka nie weszła jeszcze cała w pole
magnet.
                    strum2 = B*10*(int)(10*t)/10*boky/(1000000*x)*bokx;
                }
                else
                {
                    sem2 = 0;
                    fi2=0;
                    c=b;
                    b=0;
                }

                xx1 = (int)(10*t1); //xx2-xx1: jednostka czasu na osi
układu współrzędnych
                xx2 = (int)(10*t);

                //RYSOWANIE WYKRESÓW


//*********************************************************************
********
                //WYKRES STRUMIENIA

                if((v*t> x)&&(c==1))
```

```
                {   //odrysowanie spadku strumienia w celu kosmet., gdy
ramka wejdzie w pole
                    g2.drawLine(0, (int)(dp2h-1),70, (int)(dp2h-1 -
0.75*dp2h));
                }
                if(v*t<=x)
                {   //rysowanie strumienia, kiedy ramka wchodzi w pole
                    g2.drawLine(xx1, (int)(dp2h-1 -
0.75*(strum1/maxStrum)*dp2h),
                        xx2, (int)(dp2h-1 -
0.75*(strum2/maxStrum)*dp2h));
                }else
                {   //rysowanie strumienia, gdy ramka cała w polu
                    g2.drawLine(70, (int)(dp2h-1 - 0.75*dp2h),xx2,
(int)(dp2h-1 - 0.75*dp2h));
                }


//*********************************************************************
********
                //WYKRES SEM
                if((v*t> x)&&(c==1)) //rysowanie wykresu SEM, kiedy w
polu
                {                    //w celu kosmetycznym
                    xx2=70;
                    xx1=70;
                    g3.drawLine(0, (int)(dp3h-1 - 0.75*dp3h),
                        70, (int)(dp3h-1 - 0.75*dp3h));
                }
                //if(v*t1> x){xx1=70;}

                //rysowanie wykresu SEM
                g3.drawLine(xx1, (int)(dp3h-1 - 0.75*c*dp3h),xx2,
(int)(dp3h-1 - 0.75*b*dp3h));



//*********************************************************************
********
                strum1 = strum2;
                t1 = t;

                dp2.repaint(); //odrysowanie panelu związanego z wykresem
strumienia
                dp3.repaint(); //odrysowanie panelu związanego z wykresem
SEM

                if(v*t> x) strum2=maxStrum; //wartość strumienia po
wejściu ramki w pole
                jLabel10.setText(""+zaokraglanie(strum2*dzielB,3)+strB);
//wyświetlanie bieżącej wartości strumienia

jLabel11.setText(""+zaokraglanie(sem2*dzielSEM,3)+strSEM); //wyświetlanie
bieżącej wartości SEM

                rys_ramka();

                try
                {
                    Thread.sleep(100);
```

```
                }
                catch (InterruptedException ex)
                {

Logger.getLogger(RamkaAplet.class.getName()).log(Level.SEVERE, null, ex);
                }
                while(parent.pauza){}
            }
            while(parent.start);
        }
}

public double zaokraglanie(double liczba,int decimalPlace)
{
    BigDecimal bd = new BigDecimal(liczba);
    bd = bd.setScale(decimalPlace, BigDecimal.ROUND_HALF_DOWN);
    return bd.doubleValue();
}

    // Variables declaration - do not modify//GEN-BEGIN:variables
    private org.me.ramka.DrawingPanel dp1;
    private org.me.ramka.DrawingPanel dp2;
    private org.me.ramka.DrawingPanel dp3;
    private javax.swing.JButton jButton1;
    private javax.swing.JButton jButton2;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel10;
    private javax.swing.JLabel jLabel11;
    private javax.swing.JLabel jLabel12;
    private javax.swing.JLabel jLabel13;
    private javax.swing.JLabel jLabel14;
    private javax.swing.JLabel jLabel15;
    private javax.swing.JLabel jLabel16;
    private javax.swing.JLabel jLabel17;
    private javax.swing.JLabel jLabel2;
    private javax.swing.JLabel jLabel3;
    private javax.swing.JLabel jLabel4;
    private javax.swing.JLabel jLabel5;
    private javax.swing.JLabel jLabel6;
    private javax.swing.JLabel jLabel7;
    private javax.swing.JLabel jLabel8;
    private javax.swing.JLabel jLabel9;
    private javax.swing.JSpinner jSpinner1;
    private javax.swing.JSpinner jSpinner2;
    private javax.swing.JSpinner jSpinner3;
    private javax.swing.JSpinner jSpinner4;
    // End of variables declaration//GEN-END:variables

    double t = 0, t1 = 0;
    double B; //wejściowe
    double V; //wejściowe
    double bokx; //wejściowe
    double boky; //wejściowe
    int intbokx;
    int intboky;
    int dp2h; //wysokość dp2
    int dp2w; //szerokość dp2
    int dp3h; //wysokość dp3
    int dp3w; //szerokość dp3
    double v = 0.1;
    double x = 0.7;
```

```
    double y = 1;
    double maxStrum = 1;
    double strum1 = 0, strum2 = 0; //wartości
    double sem2 = 0;
    double fi1,fi2=0;
    double czas;
    double deltat; //krok czasowy
    double liczba; //do wyświetlania wartości na skali
    double wartB;
    double wartSEM;
    double dzielB = 1;
    double dzielSEM =1;
    String strB;
    String strSEM;
    int b;
    int c;
    boolean start = false;
    boolean pauza = false;
    int xx1 = 0, xx2 = 0;

    double tmp = 0;
}
```

8.6. Załącznik 6.

Kod źródłowy *Ramka obracająca się w polu magnetycznym*:

Plik *BufferedAplet.java*:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package paczka;
import java.applet.Applet;
import java.awt.*;
/**
 *
 * @author Krzysztof
 */
public class BufferedApplet extends Applet{
    protected boolean podwojne_buforowanie=true;
    private Image bufor=null;
    public BufferedApplet()
    {
    super(); //wywolanie konstruktora klasy Applet
    }
    @Override
    public void update(Graphics g)
    {
        Rectangle dim=this.getBounds();
        int w=dim.width, h=dim.height;
        if (podwojne_buforowanie)
        {
            if (bufor==null) bufor=this.createImage(w,h);
            Graphics bg=bufor.getGraphics();
            bg.clearRect(0,0,w,h);
            bg.setClip(0,0,w,h);
```

```
            this.paint(bg);
            g.drawImage(bufor,0,0,null);
            bg.dispose();
        }
        else
        {
            g.clearRect(0,0,w,h);
            paint(g);
        }
    }
}
```

Plik *NewApplet.java*:

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package paczka;

import java.awt.*;
import java.awt.event.ActionEvent;
import java.util.*;
import javax.swing.JButton;
import javax.swing.JSlider;

/**
 *
 * @author Administrator
 */
public class NewApplet extends BufferedApplet {
    int i =0;
    int nr=1;
    double a=0;
    int yaxis=0;
    int y1axis=0;
    int rr=5;
    int r=70;
    int mx=80;
    int my=60;
    int height=150;
    int width=200;
    double f=1;
    boolean start=true;
    int c;
    int ab=1;
    JButton przycisk1;
    JSlider slider;
    /**
     * Initialization method that will be called after the applet is
loaded
     * into the browser.
     */
    @Override
    public void init() {
        try {
            java.awt.EventQueue.invokeAndWait(new Runnable() {
                public void run() {
                    setSize(540, 300);
                    setLayout(new FlowLayout(FlowLayout.LEFT,40,250));
```

```
                    przycisk1 =  new JButton("=START=");
                    przycisk1.addActionListener(new
java.awt.event.ActionListener() {
                    public void
actionPerformed(java.awt.event.ActionEvent evt) {
                        przycisk1ActionPerformed(evt);
                    }
                    });
                    add(przycisk1);
                    slider = new JSlider();
                    slider.setMinimum(1);
                    slider.setMaximum(6);
                    slider.setBackground(new Color(0,150,0,255));
                    add(slider);
                    setBackground(new Color(0,150,0,255));
                }
            });
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    private void przycisk1ActionPerformed(ActionEvent evt) {
        if (start)
        {
            start=false;
            Uruchom();
            przycisk1.setText("PAUSE");
        }else
        {
            start=true;
            przycisk1.setText("=START=");
        }
    }

    @Override
    public void paint(Graphics g)
    {
        int x, x0,xV0;
        double y0, y,yV,yV0, A,  t, offset;
        A = (double) height / 4;
        offset = (double) height / 2;
        x0 = 0;
        xV0=0;
        y0 = offset;
        yV0=offset;
        super.paint(g);
        g.setFont(new Font("Comic Sans MS",Font.PLAIN,12));
        g.drawString("Regulator prędkości obrotu ramki", 170, 252);
        int [] xPoints={0,0,0,0,0,0,0,0};
        if(nr==1)
        {
            xPoints[0]=mx-yaxis/4;
            xPoints[1]=mx+140+yaxis/4;
            xPoints[2]=mx+140-y1axis/4;
            xPoints[3]=mx+170+y1axis/4;
            xPoints[4]=mx+170+y1axis/4;
            xPoints[5]=mx+140-y1axis/4;
            xPoints[6]=mx+140-yaxis/4;
            xPoints[7]=mx+yaxis/4;
        }else
```

```
        {
            xPoints[7]=mx-yaxis/4;
            xPoints[6]=mx+140+yaxis/4;
            xPoints[5]=mx+140-y1axis/4;
            xPoints[4]=mx+170+y1axis/4;
            xPoints[3]=mx+170+y1axis/4;
            xPoints[2]=mx+140-y1axis/4;
            xPoints[1]=mx+140-yaxis/4;
            xPoints[0]=mx+yaxis/4;
        }
        int [] yPoints =
{my+Math.abs(yaxis),my+Math.abs(yaxis),my+65+Math.abs(y1axis),
        my+65+Math.abs(y1axis),my+75-Math.abs(y1axis),my+75-
Math.abs(y1axis),
        my+140-Math.abs(yaxis),my+140-Math.abs(yaxis)};

        g.setColor(Color.red);
        g.fillRect(mx-40, 220, 220, 15); //pasek czerwony
        int [] a2={mx-10,mx+150,mx+180,mx-40};
        int [] b2={my+125,my+125,my+160,my+160};
        g.fillPolygon(a2,b2,4);

        g.setColor(Color.blue);
        g.fillRect(mx-40, 20, 220, 15); //pasek niebieski
        int [] a1={mx-10,mx+150,mx+180,mx-40};
        int [] b1={my+10,my+10,my-25,my-25};
        g.fillPolygon(a1,b1,4);

        g.setColor(Color.black);
        g.drawRect(mx-40, 20, 220, 15);
        g.drawRect(mx-40, 220, 220, 15);
        g.drawPolygon(a1,b1,4);
        g.drawPolygon(a2,b2,4);
        g.setColor(Color.white);
        g.drawString("N", 140, 32);
        g.drawString("S", 140, 232);

        //przewody
        g.setColor(new Color(254,248,180));
        g.fillRect(255, 125, 100, 3);
        g.fillRect(255, 133, 100, 3);

        //podłączenia
        g.setColor(new Color(40,40,40));
        g.fillRoundRect(280, 121, 10, 9,1,1);
        g.fillRoundRect(280, 132, 10, 9,1,1);

        //linie pola
        g.setColor(Color.white);
        g.fill3DRect(75, 56, 1, 140,true);
        g.fill3DRect(105, 56, 1, 140,true);
        g.fill3DRect(135, 56, 1, 140,true);
        g.fill3DRect(165, 56, 1, 140,true);
        g.fill3DRect(195, 56, 1, 140,true);
        g.fill3DRect(225, 56, 1, 140,true);
        g.drawLine(75,140,73,135);
        g.drawLine(75,140,77,135);
        g.drawLine(105,140,103,135);
        g.drawLine(105,140,107,135);
        g.drawLine(135,140,133,135);
        g.drawLine(135,140,137,135);
```

```
        g.drawLine(165,140,163,135);
        g.drawLine(165,140,167,135);
        g.drawLine(195,140,193,135);
        g.drawLine(195,140,197,135);
        g.drawLine(225,140,223,135);
        g.drawLine(225,140,227,135);

        g.setColor(Color.orange);
        g.drawPolygon(xPoints, yPoints, 8); //ramka
        g.setColor(Color.DARK_GRAY);
        g.fillRoundRect(170+mx, 60+my, 10, 20, 2, 2); //klocek

        //linie pola
        g.setColor(Color.white);
        g.fill3DRect(50, 41, 1, 175,true);
        g.fill3DRect(90, 41, 1, 175,true);
        g.fill3DRect(130, 41, 1, 175,true);
        g.fill3DRect(170, 41, 1, 175,true);
        g.fill3DRect(210, 41, 1, 175,true);
        g.fill3DRect(250, 41, 1, 175,true);
        g.drawLine(50,160,48,155);
        g.drawLine(50,160,52,155);
        g.drawLine(90,160,88,155);
        g.drawLine(90,160,92,155);
        g.drawLine(130,160,128,155);
        g.drawLine(130,160,132,155);
        g.drawLine(170,160,168,155);
        g.drawLine(170,160,172,155);
        g.drawLine(210,160,208,155);
        g.drawLine(210,160,212,155);
        g.drawLine(250,160,248,155);
        g.drawLine(250,160,252,155);
        //miernik
        g.setColor(new Color(48,48,48));
        g.fillRoundRect(290+x0, (int) y0-65, 240,190,3,3);
        g.setColor(new Color(40,40,40));
        g.fillRoundRect(310+x0, (int) y0+125, 40,20,2,2);
        g.fillRoundRect(470+x0, (int) y0+125, 40,20,2,2);

        g.setColor(Color.gray);
        g.fillRoundRect(305, 21,211,169,3,3); //ekran
        g.setColor(Color.green);
        g.fillRect(309, 24,203,163);
        //wykres

        g.setColor(Color.black);
        g.drawLine (311+width/2, 25, 311+width/2,30+ height); //linia
pionu
        g.drawLine (310+x0, 31+(int) y0, 310+width,31+ (int) y0); //linia
poziomu
        for (int j=0;j<11;j++)
            g.drawLine (210+width/2+j*20, 25, 210+width/2+j*20,35+
height); //linia pionu
        for (int j=0;j<9;j++)
            g.drawLine (310+x0, (int) y0-50+j*20, 310+width,(int) y0-
50+j*20); //linia poziomu
        c=slider.getValue();
        for (x=0; x<width; x+=1) {
              t = (double) x /(double) width;
              y = offset - A * Math.sin (Math.PI * f*
t+Math.PI/(120)*i*f-Math.PI*(f-1)/2);
```

```
              yV = offset - A * Math.cos(Math.PI * f*
t+Math.PI/(120)*i*f-Math.PI*(f-1)/2)*c/3;
              g.setColor(Color.red);
              g.drawLine (310+x0, 30+(int) y0, 310+x, 30+(int) y);
              g.setColor(new Color(0,100,0));
              g.drawLine (310+x0, 30+(int) yV0, 310+x, 30+(int) yV);
              x0 = x;
              y0 = y;
              yV0=yV;
       }

       //ekranik
        g.setColor(new Color(20,20,20));
        g.fillRoundRect(355,170, 110,40,2,2);
        g.setColor(Color.white);
        g.fillRoundRect(360,175, 100,30,2,2);
        g.setColor(Color.black);
        g.drawString("sin(a)", 365, 195);
        g.drawString("V", 425, 195);
        g.setColor(Color.red);
        g.fillOval(400, 185, 10,10);
        g.setColor(Color.green);
        g.fillOval(440, 185, 10,10);
        g.setFont(new Font("Comic Sans MS",Font.PLAIN,10));
        g.drawString("a - kąt między prostopadłą", 400, 232);
        g.drawString("do powierzchni ramki,", 400, 242);
        g.drawString("a liniami pola magnet.", 400, 252);
        g.drawString("V - indukowana SEM", 400, 265);
    }

    public void Uruchom()
    {
        Timer latanie = new Timer();
        latanie.schedule(new TimerTask(){

            @Override
            public void run() {
                i++;
                f=slider.getValue();
                a=Math.sin(Math.PI/120*i*f);
                yaxis=(int)(r*a);
                y1axis=(int)(rr*a);
                if (a==1)
                    nr=0;
                else
                    if (a==-1)
                        nr=1;
                repaint();
                if (start==true) this.cancel();
            }
        }, 0,50);
    }

}
```

8.7. Załącznik 7.

Kod źródłowy *Elektryzowanie przez tarcie*:

Plik *BufferedApplet.java*: zawartość jak w załączniku 6.

Plik *Obrazek.java*:

```
/*
 * Obrazek.java
 *
 * Created on 28 luty 2009, 13:35
 */

package ruch;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Image;
import java.util.Timer;
import java.util.TimerTask;

/**
 *
 * @author  Krzysztof
 */


public class Obrazek extends BufferedApplet {
    public static final int A=10;
    int i=0,a=0,b=0,c=0,p=0;
    boolean d=true;
    Image img1,img2,plus,minus;
    /** Initializes the applet Obrazek */

    @Override
    public void init() {
        try {
            java.awt.EventQueue.invokeAndWait(new Runnable() {
                public void run() {
                    initComponents();
                    laduj();
                    setBackground(new Color(0,150,0));
                }
            });
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }

    public void laduj()
    {
        try
        {
        img1 = getImage(getDocumentBase(), "linijka.gif");
        img2 = getImage(getDocumentBase(), "image8.gif");
        plus = getImage(getDocumentBase(), "+.gif");
        minus = getImage(getDocumentBase(), "-.gif");
        }catch(Exception ex)
        {
            System.out.println("Nie udalo sie zaladowac obrazków");
        }
    }

    @Override
    public void paint(Graphics g)
    {
        super.paint(g);
```

```
    //linijka
    g.drawImage(img1, A, A+10,this);
    //plusy i minusy (elektrony i protony) na linijce
    g.drawImage(minus, A+40, A+170, this);
    g.drawImage(minus, A+120, A+110, this);
    g.drawImage(minus, A+200, A+50, this);
    g.drawImage(minus, A+70, A+115, this);
    g.drawImage(minus, A+150, A+60, this);
    g.drawImage(plus, A+90, A+130, this);
    g.drawImage(plus, A+160, A+80, this);
    g.drawImage(plus, A+30, A+140, this);
    g.drawImage(plus, A+110, A+85, this);
    g.drawImage(plus, A+190, A+30, this);
    if(d)
    {
        //futerko
        g.drawImage(img2, A+200+b-i, A+80-b, this);
        //plusy i minusy (elektrony i protony) na futerku
        if (a<2)
        {
            g.drawImage(minus, A+215+b-i, A+145-b, this); // minus 1
            g.drawImage(minus, A+265+b-i, A+100-b, this); // minus 2
        } else
        {
            g.setColor(Color.yellow);
            g.fillOval(A+212-i, A+137, 22, 22); // minus 1
            g.fillOval(A+262-i+c, A+92-c, 22, 22); // minus2
            g.drawImage(minus, A+215-i, A+145, this);  // minus 1
            g.drawImage(minus, A+265-i+c, A+100-c, this); // minus 2
        }
        //na stałe na szmatce
        g.drawImage(plus, A+235+b-i, A+145-b, this); // plus 1
        g.drawImage(plus, A+285+b-i, A+105-b, this); //plus 2
        g.drawImage(plus, A+240+b-i, A+120-b, this); //plus 3
        g.drawImage(minus, A+260+b-i, A+130-b, this); //minus 3
    }else
    { //powrot szmatki
        g.drawImage(img2, A+200+b-i+p, A+80-b, this); //szmatka
        g.setColor(Color.yellow);
        g.fillOval(A+212-i, A+137, 22, 22); //minus 1
        g.fillOval(A+262-i+c, A+92-c, 22, 22); // minus 2
        g.fillOval(A+230+b-i+p, A+140-b, 22, 22); // plus 1
        g.fillOval(A+280+b-i+p, A+100-b, 22, 22); // plus 2

        g.drawImage(minus, A+215-i, A+145, this);  // minus 1
        g.drawImage(minus, A+265-i+c, A+100-c, this); // minus 2
        g.drawImage(minus, A+260+b-i+p, A+130-b, this); //minus 3
        g.drawImage(plus, A+235+b-i+p, A+145-b, this); // plus 1
        g.drawImage(plus, A+285+b-i+p, A+105-b, this); // plus 2
        g.drawImage(plus, A+240+b-i+p, A+120-b, this); //plus 3
    }
}


public void Uruchom()
{
    Timer latanie = new Timer();
    latanie.schedule(new TimerTask(){

        @Override
        public void run() {
```

```
            i++;
            if(i>150)
            {
                this.cancel();
                pocieranie();
            }
            repaint();
        }

    }, 0,5);
}

public void pocieranie()
{
    a++;
    Timer potrzyj = new Timer();
    potrzyj.schedule(new TimerTask(){

        @Override
        public void run() {
            b++;
            if(a==2) c+=2;
            if(b>=30)
            {
                this.cancel();
                pocieraniedol();
            }
            repaint();
        }

    }, 0,30);
}

public void pocieraniedol()
{
    if (a!=3)
    {
    Timer trzyj = new Timer();
    trzyj.schedule(new TimerTask(){

        @Override
        public void run() {
            b--;
            if(b<1)
            {
                this.cancel();
                pocieranie();
            }
            repaint();
        }

    }, 0,30);
    }
    if(a==3)
    {
        d=false;
        powrot();
    }
}

public void powrot()
```

```
    {
        Timer powr=new Timer();
        powr.schedule(new TimerTask(){

            @Override
            public void run() {
                p+=2;
                if (p>100)
                {
                    jButton1.setEnabled(true);
                    this.cancel();
                }
                repaint();
            }

        }, 0, 20);
    }
    /** This method is called from within the init() method to
     * initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is
     * always regenerated by the Form Editor.
     */
    // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-
BEGIN:initComponents
    private void initComponents() {

        jButton1 = new javax.swing.JButton();
        jScrollPane1 = new javax.swing.JScrollPane();
        jTextArea1 = new javax.swing.JTextArea();

        setBackground(java.awt.Color.lightGray);
        setForeground(java.awt.Color.white);

        jButton1.setFont(new java.awt.Font("Comic Sans MS", 1, 14));
        jButton1.setText("START");
        jButton1.setPreferredSize(new java.awt.Dimension(40, 27));
        jButton1.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                jButton1ActionPerformed(evt);
            }
        });


jScrollPane1.setHorizontalScrollBarPolicy(javax.swing.ScrollPaneConstants
.HORIZONTAL_SCROLLBAR_NEVER);

jScrollPane1.setVerticalScrollBarPolicy(javax.swing.ScrollPaneConstants.V
ERTICAL_SCROLLBAR_NEVER);

jScrollPane1.setViewportBorder(javax.swing.BorderFactory.createLineBorder
(new java.awt.Color(0, 0, 0)));

        jTextArea1.setBackground(new java.awt.Color(137, 223, 137));
        jTextArea1.setColumns(20);
        jTextArea1.setFont(new java.awt.Font("Comic Sans MS", 1, 12));
        jTextArea1.setLineWrap(true);
        jTextArea1.setRows(5);
        jTextArea1.setText("Elektryzowanie przez tarcie polega na
przepływie elektronów z jednego \nciała do drugiego podczas pocierania
ich o siebie. W ten sposób jedno ciało \nposiada nadmiar ładunku
dodatniego, drugie natomiast ładunku ujemnego. \nPocieranie zwiększa
```

```
liczbę punktów styczności pomiędzy ciałami. Przykładem \ntego sposobu
eiektryzowania może być pocieranie końca plastikowej linijki futrem. \nW
punktach ich styczności część ładunku ujemnego przechodzi z futra na
linijkę \nnaruszając obojętność elektryczną obu materiałów.");
        jTextArea1.setWrapStyleWord(true);
        jTextArea1.setMargin(new java.awt.Insets(4, 10, 2, 2));
        jScrollPane1.setViewportView(jTextArea1);

        javax.swing.GroupLayout layout = new
javax.swing.GroupLayout(this);
        this.setLayout(layout);
        layout.setHorizontalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(layout.createSequentialGroup()
                .addContainerGap()

.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TR
AILING)
                    .addComponent(jButton1,
javax.swing.GroupLayout.PREFERRED_SIZE, 96,
javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 501,
javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(21, Short.MAX_VALUE))
        );
        layout.setVerticalGroup(

layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
layout.createSequentialGroup()
                .addContainerGap(284, Short.MAX_VALUE)
                .addComponent(jButton1,
javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.PREFERRED_SIZE)

.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(jScrollPane1,
javax.swing.GroupLayout.PREFERRED_SIZE, 141,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
    }// </editor-fold>//GEN-END:initComponents

private void jButton1ActionPerformed(java.awt.event.ActionEvent evt)
{//GEN-FIRST:event_jButton1ActionPerformed
    i=0; a=0; b=0;c=0;p=0;d=true;
    jButton1.setEnabled(false);
    this.repaint();
    Uruchom();
}//GEN-LAST:event_jButton1ActionPerformed


    // Variables declaration - do not modify//GEN-BEGIN:variables
    private javax.swing.JButton jButton1;
    private javax.swing.JScrollPane jScrollPane1;
    private javax.swing.JTextArea jTextArea1;
    // End of variables declaration//GEN-END:variables
}
```